An Agent-based Semantic Search Engine for Scalable Enterprise Applications*

Andrea Passadore[1], Alberto Grosso[1], Antonio Boccalatte[1]

[1] University of Genova, DIST, Via Opera Pia 13, 16145 Genova, Italy
{passa, agrosso, nino}dist.unige.it

Abstract. In this paper we present AgentSeeker: a multi-agent platform aimed to index local or online documents, with the support of ontologies which describe the application domain and the competences the user is referring to, during his query session. AgentSeeker is a flexible and scalable solution mainly devoted to enterprise applications where electronic knowledge bases are particularly important for their business activity. An Ontology Agent is devoted to manage semantic representations of the enterprise domain, organizing the results of a user's query, according to the concepts which represent the relevant entities in the company business.

Keywords: ontology, search engine, multi-agent system, web site, document.

1 Introduction

The Information Age leads to us various benefits and comforts, encouraging our thirst for knowledge, helping us at work, and gladdening us during our leisure time. On the other hand, the bits which encode the information are so much that we are lost in a sea of electronic data. The orientation is so hard that oftentimes we lose documents both in the little pond of our personal hard disks and in the boundless ocean of Internet.

Search engines represent a saving compass which enables us to find an Internet page winnowing the whole network or to find a personal document through a desktop application which parses private files.

Usual search engines denote an intuitive behaviour: they store the textual content of the parsed documents in a database and they return an ordered list of files containing the keywords suggested through a user's query. Proper algorithms calculate the rank for every hit and, according to this evaluation, the search engines display first the most relevant pages. In spite of this solution, it is a user's experience that sometimes the search engine gives a completely wrong page link due to a misunderstanding of the meaning of the keyword or neglects a page link which does not explicitly contain the given term but it is anyway relevant.

The aim of *AgentSeeker*, the search engine presented in this paper, is to make the document retrieval a more intelligent process, finding texts which are semantically bind to the user's query. In order to achieve this goal, two aspects of AgentSeeker are

* AgentSeeker is partially based on the Master Thesis of Fabrizio Scaglione. Thank you Fabrizio for your accurate work.

relevant: *software agents* and *ontologies*. Based on a multi-agent platform, AgentSeeker is a scalable and flexible solution which can be adapted to different contexts. AgentSeeker agents are able to manage ontologies in order to constitute a semantic tool for helping users to not lose their course during a search session in the electronic ocean.

Even if AgentSeeker is not designed for competing with the giant search engines as Google or Yahoo, it is aimed both to index Internet pages and local files and it is especially focused to enterprise contexts where the value of the digital information is particularly high. Enterprises entrust more and more often their documents to digital storage devices and base their knowledge on electronic sources. AgentSeeker helps users to retrieve these documents, tuning the response on the basis of the company's application domain or the user's skills, through the definition or the import of a specific ontology. A particular kind of agent is able to manage these ontologies, integrating the user's queries with semantically related words, discovered through the analysis of concepts relations, specializations, and synonyms.

AgentSeeker is able to manage different amount of textual documents: from a little *corpus* on a single file server, to a big collection scattered on a network. The agent roles involved in AgentSeeker are designed in order to operate in a variable amount of instances and to interact with peers in a circumscribed environment (a single PC) alike a distributed platform federation.

After a brief survey on the state-of-the-art in the search engines field (section 2), the multi-agent platform is shown in details (section 3). Section 4 is then completely aimed to the semantic layer of the application, showing how ontologies support the user's interactions with the system. Conclusions and future works will follow.

2 Search engines: the state-of-the-art

To google is a neologism born to specify the act of performing a web search on a search engine named *Google*. Considering its indisputable supremacy in respect with the direct competitors, we introduce Google, in order to briefly explain the traditional search engines. Since AgentSeeker is both an *online* and an *offline* search engine, this state-of-the-art survey distinguishes between online applications and offline tools (aimed to the indexing of local documents).

Google is now a pharaonic effort to index not only textual documents on the web, but also images, videos, papers, news, etc. Founded in the 1998 by two researchers of the Stanford University, Google bases its architecture on few and very clear ideas: *scalability*, *redundancy*, and *pragmatism*. A distributed environment (named *Google File System* (GFS) [1]) of 450000 computers grouped in clusters ensures scalability and redundancy in order to follow the constant growth of online web pages. A large amount of data is processed by applications developed following the *MapReduce* paradigm [2]. This model consists in two functions *map* and *reduce* and a tree of computational nodes. *Map* and *reduce* respectively allow a node to split a problem into sub-problems submitted to child nodes and to collect the results.

If we recall the aforementioned distinction between the ocean of Internet and the local pond represented by our hard disks, Google provides also an offline engine

(named *Google Desktop*) which indexes local files as *pdf*, textual files, and e-mails. It stores them in a local index which is constantly updated.

Another horizon for an offline search engine is to operate in an intranet context, like the computers of an enterprise: in this scenario the document *corpus* can be scattered on several file servers or shared folders of employees' hard disks. There exist several industrial solutions (generally called *enterprise search engines*, or *intranet search engines*) aimed to address the functionalities of the usual online search engines to the internal network. *Vivisimo* is an interesting tool also for its innovative architecture (see paragraph 2.1) which represents a significant improvement in the search engines panorama. Other significant tools are *Northern Light*, which includes also a sort of document classification based on taxonomies; *Search Engine Studio*; *Noematics Reflexion*; etc.

All things considered, they are standard applications with well consolidated technologies and they do not represent (apart Vivisimo) particular efforts to refine the behaviour of search engines. On the other hand, they show the clear evidence that the problem of managing collections of textual documents is a sensible aspect which involves not only Internet surfers, but also entire enterprises.

2.1 Towards smart search engines

Oftentimes, computer users clash with the efficient but sometimes not very intelligent behaviour of machines. Also search engines can show disappointing behaviours furnishing results which are not in line with user's expectations.

For this reason there exist several studies with the main goal to increase search engine performances. As introduced before, Vivisimo [3] is a significant example of an offline enterprise search engine (used by big enterprises as *Airbus*, *Cisco*, *P&G*) which introduces a form of intelligence, by providing results automatically classified in hierarchical clusters.

Other mature solutions available online are *iBoogie*, *SnakeT*, *KWMAP*, and *Exalead*.

2.2 Ontological search engines and agent-based solutions

Since AgentSeeker is an application based on the multi-agent paradigm and on ontological representations, before to show its architecture we proceed with an evaluation of similar existing solutions. At now, these solutions are essentially proposed by the research community, with no contributions from the industrial world.

Regarding multi-agent solutions, a very common approach is to use a multi-agent platform as a *meta-search engine*, namely an interface among the user and a set of well-known online engines. For example, *ProFusion* [4] executes agents able to connect to *AltaVista*, *Yahoo*, and *Excite*; other agents are responsible of merging the results or monitoring the status of the aforementioned search engines. Another meta-search engine is *MAGI* [5] which differs from ProFusion for its re-ranking technique exploited in order to sort the results coming from the different sources.

Regarding meta-search engines, another proposal is done in [6] and it seems the closest to AgentSeeker because it uses ontologies in order to represent the query through a concept network, instead of a mere string. Then, dedicated agents parse the pages suggested by Google and construct the relative concept networks, analyzing the extracted text. Finally, the meta search engine returns first the pages the engine considers significant, namely those pages which have a concept network compatible with the query one. *MAIQS* [7] is the only known multi-agent platform which implements an enterprise search engine, limited to small teams composed of 10-20 employees. Every user can share his documents hosting on his machine an indexing agent that builds a local index. A mechanism of query propagation ensures that a search session is extended to the whole knowledge base of the team. [6] is the only example of productive cooperation among agents and ontologies in the search engine field. Nevertheless, there exist some proposals of advanced search engines which exploit the powerfulness of the Semantic Web and in particular of ontologies. *EKOSS* [8] is a project aimed to the sharing of knowledge in form of deliverables containing papers, lecture materials, computational models, or multi-media files. Every peer user of EKOSS classifies these resources by using a sort of ontological tags, coming from conceptual representations which can be explored by the user in order to access the knowledge in a semantic way. *MOSSE* [9] is a proposed solution which supports document classification based on the first 15 categories of the directory DMOZ (www.dmoz.org) and a sort of query expansion based on first senses and hypernyms of WordNet. Although MOSSE is an ambitious solution, it has been implemented only in a minimal part. [10] is focused on cataloguing of wide video archives with meta-tags derived from ontologies. The system is essentially based on two main features: the retrieval of videos on the basis of meta-tags and the analysis of frames, in order to extract basic patterns which allow the comparison of two images.

Meta-tagging seems to be a very common approach, in order to add semantics to textual and multi-media files (see also [11] or [12]), but some considerations suggest to us to pursue another way, as described in the following section.

3 AgentSeeker: a federation of multi-agent platforms

AgentSeeker has been developed by following few basic principles: scalability, flexibility, and careful management of textual documents. As seen in the previous section there is a concrete interest in enterprise search engines and a trend of increasing their performances with a smart management of the results.

In order to consider documents not only as mere aggregates of characters and numbers but also as knowledge with a precise meaning, several solutions use meta-tags in order to semantically describe their content. Nevertheless, if we consider as source of information Internet or large local document *corpi*, the tagging of these resources become very complicated due to the impossibility of modifying a file or to the objective difficulty to manually catalogue thousands of documents. For this practical reason AgentSeeker has only the textual content available, and the ontologies are used to describe the knowledge of the user, his skills, and his expectations in order to apply them during the document search.

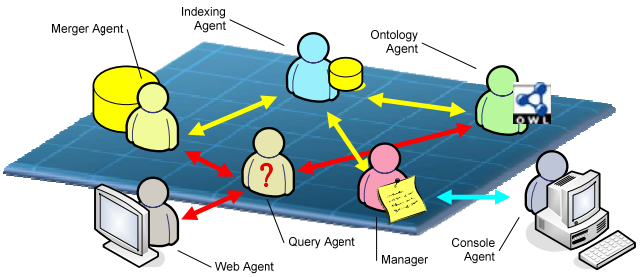

Fig. 1. the agent roles in AgentSeeker

3.1 Implementing AgentSeeker with AgentService

AgentSeeker is essentially based on AgentService [13], a framework for the development and execution of multi-agent systems implemented in the C# programming language and using the Microsoft .NET libraries.

AgentService was born in order to offer the possibility of developing software agents in an industrial context, where the .NET framework and C# are appreciated. Anyway, AgentService is compatible with the open source libraries of the Mono Project and it is successfully ported on Linux-based platforms.

We avoid a full introduction of the AgentService framework (for further details see [13] or [14]) and we focus only on few concepts which will be useful in order to comprehend the AgentSeeker architecture. AgentService is a framework inspired by the FIPA specification [15] for multi-agent systems. Following this specification, AgentService supports software agents furnishing a runtime environment which manages the whole life-cycle of an agent, schedules its activities, dispatches its messages transporting them to the destination, and publishes its services in order to share them with peers (a sort of *yellow pages* service). Moreover, AgentService is a modular architecture highly customizable in order to meet user's requirements; for example the user can customize the scheduling system, the messaging module, or create his own module for a new kind of service.

An AgentService platform can be placed in an ecosystem where its agents can interact in an easy and transparent way with peers resident in platforms running on remote computers or on mobile devices connected through a wireless network; they can also interact with external applications masked as agents through a web service. In this distributed context, agents can share their services by using a distributed yellow pages service which comprises all the federation. The services of agents are essentially their capabilities and are implemented as concurrent behaviours. The behaviour, together with the knowledge object, is a constitutional element of the AgentService agent model. *Behaviours* can be implemented as concurrent threads and embody the business logic of the agent. *Knowledge objects* are fully customizable data structures which represent the knowledge base of the agent. They are shared and accessed concurrently by the running agent behaviours. Both behaviours and knowledge objects can be instanced in multiple copies, also in the same agent.

3.2 The AgentSeeker architecture

AgentSeeker is a community of agents which are specialized in specific roles interacting in a coordinated way. Fig 1 shows the different roles and their interactions. There are two kinds of agents: internal agents which are usual AgentService entities equipped with behaviours and knowledge objects, and external agents, namely external programs in form of web applications or desktop client applications which act like AgentService agents in order to easily interact with the rest of the platform.

The indexing agent

The Indexing Agent (IA) is the core of the system. It is the agent able to download files from the web or from intranet repositories and to extract the textual content. At now, the IA is able to parse files of the following types: *html*, *pdf*, *doc*, *ppt* (powerpoint), and *xls* (excel). Parsing hypertexts, the agent extracts also the hyperlinks and distinguishes from internal links (namely pages which belongs to the same site) and external links (pages of other sites). In case of external page, the IA collects the link in a list which will be sent to the manager agent (its features are described below).

When an IA has been created, it advertises its competence (namely the indexing ability) through the yellow pages service. When the agent receives a job (a web site link to visit), it deregisters itself from the yellow pages in order to receive no other jobs. Once the current indexing session is finished, the IA registers itself again.

For each indexing session, the agent maintains its own database where stores information extracted from the parsed files (*path*, *content*, *title*, etc.). The local database is based on *.NET Lucene*: the C# porting of a well-known *Apache Foundation* java project named Apache Lucene [16]. Essentially developed to store textual contents and to operate queries on them, Lucene is a scalable solution that allows the implementation of large architectures. To confirm the quality of the Lucene solution, it has been used in many famous projects as *mediaWiki* (the engine of Wikipedia), *Beagle* (based on the .NET porting of Lucene), *DPSpace* (a project managed by MIT and HP labs), *LjFind* (an indexing engine for 110.000.000 blogs), *Eclipse* (the development framework for Java uses Lucene to index its guide), and DMOZ (an open directory for web sites).

The session index is then shared with the merger agent, which is aimed to manage a central index where are merged the various session indexes coming from IA instances.

Merging the indexing agents results

The task of the Merger Agent (MA) is to collect the results of IAs and to merge them in a central index (based on Lucene). Exploiting the features of the Apache Foundation's project it is possible to merge partial indexes avoiding the replication of records and applying an optimization of the database, through data compression. The MA is also devoted to practically execute a query coming from the rest of the platform, in particular from the query agent.

The Query Agent

The Query Agent (QA) receives from the outside a textual query. Possible senders could be the Web Interface Agent or the Administration Console Agent, two external agents which directly interact with a user. The QA maintains the list of MAs and submits the query to each of them; once every reply has been received, the QA collects and orders the results on the basis of the ranking expressed by Lucene.

An important interaction of the QA is the conversation with the Ontology Agent in order to enrich the query with related words (see section 4). The agent sends the user's query to the Ontology Agent and receives an expanded query which will be submitted to the mergers.

The Ontology Agent: a repository for semantic representations

The Ontology Agent (OA) is the keeper of the knowledge of the system. Its functionalities will be fully described in the next section but, as an introduction, the OA is essentially able to read ontologies in the OWL language, thanks to the libraries *SemWeb* and *Linq to RDF*. The OA extracts the described concepts and finds the relations among them. On the basis of this information, the OA extends the query sent by the QA, during a user's session.

Another feature of the OA is the classification of the document content. As described in section 4, the ontologies contained in the repository are considered as simple taxonomies and used to classify documents on the basis of the term occurrences. This particular service is used by the IA during its indexing sessions, which then receives an estimate of the arguments dealt in the examined text.

An external agent managing user's interactions

From the user's point of view, AgentSeeker is a simple web application with a look-and-feel similar to the usual search engines. Developed as an ASP.NET application, the web form hides, in reality, a sort of agent which, through a web service interface, contacts the remote AgentSeeker installation in order to submit a query. The life-cycle of this agent is tied with the user's session; every user has his own agent which helps him to interact with the platform. The choice of implementing the web application as an agent simplifies the development of the whole system and integrates the user's interface with the rest of the platform. In order to exploit all the features of AgentSeeker, the user has only to submit a query and select the ontology (namely the argument) he wants. Furthermore, he can import an ontology from the web suggesting its URI. This feature makes AgentSeeker a very flexible system, because it is not calibrated only on built-in ontologies, but it is open to every OWL-based file. Finally, the user can select the policy for query extension (see Section 4).

Administration console

Similar to the previous one, another pseudo-agent runs behind an administration console which allows administrators to manage AgentSeeker. For example, an Administrator can submit a new web site to index, set the standby time for the platform, or he

can directly shoot down the platform, stopping safely every agent instance. He can also monitor the status of the platform, namely the agent health, the progression of the indexing tasks, etc.

The manager role: a platform orchestrator

The manager agent is a sort of supervisor which coordinates the activity of the other agents. In particular the manager has a knowledge object containing the list of web sites (on shared folders) to parse. This list can be increased by adding new sites received from the external agent representing the administration console and by receiving new links discovered by the IAs. In presence of new links to visit, the manager searches for a free IA, consulting the yellow pages. Due to the fact that the yellow pages are distributed across the whole federation, the manager is able to find free agents running also on remote computers. The computational load is then naturally balanced on every machine and every agent.

The manager is also responsible of the standby of the whole federation, following up the specific user's command or an expired timeout.

3.3 The AgentSeeker federation

The simplest deployment of AgentSeeker consists in a single platform (in execution on a single computer) with single instances of each agent role. A manager sends jobs to the unique IA, which parses each web site (or folder), classifying every page with the help of the OA. The MA collects the results of the IA, while the QA directly speaks with the external agent behind the web application and with the OA in order to extend the query. A console agent manages the platform.

If the computer has enough resources, the platform administrator could create different instances of the IA in order to process in parallel several jobs. This is particularly useful if the CPU is multi-core, considering also that every IA alternates processing time and downloading of documents.

In case of large amount of textual documents to index, it could be useful to add further computational resources. A new computer is then connected to the first one, a new AgentService platform is installed and new IAs are instanced. The unique manager agent has now at its disposal new IAs which can be contacted through the distributed yellow pages, in a completely transparent way with no complications due to the distributed environment.

Now, with different instances of IAs, only one MA could be not enough. In this case, a new MA can be instanced and the IAs can be instructed in order to refer to a particular MA. With multiple MAs, the QA can submit the query in parallel and then compose the incoming results.

If the catchment area is wide, the federation could be integrated with several instances of query and ontology agents in order to serve different users at the same time.

At this point the scenario can be configured in various ways, with resources totally dedicated to a single type of agent, and mixed platforms with various agent roles. The single computer platform is now spread on a distributed network, in a totally transpar-

ent way from the point of view of the AgentSeeker developer and especially of the system administrator. Furthermore, new computational resources and agent instances can be added or removed dynamically during the AgentSeeker execution.

4 The semantic layer of AgentSeeker

AgentSeeker tries to propose a contribution to the improvement of search engine techniques by introducing the use of ontologies among its agents. As introduced in advance, we use ontologies in order to simply model the discourse domain to which the user is referring during the query submission. AgentSeeker is designed in order to support every kind of ontology expressed in OWL.

Considering that AgentSeeker is, at the present moment, a functioning prototype, we have implemented different policies which drive the user during his searching sessions.

A priori classification

A first policy is to classify documents on the basis of the distinctions made by the IA with the help of the OA. During the indexing session, for each extracted text, the ontology agent estimates its affinity with the topics described in the ontologies stored in the AgentSeeker repository. Every record stored in Lucene has a field where are included the URIs of the ontologies directly supported by AgentSeeker and a measure of the affinity, in term of percentage of words of the document which are also contained in the ontology. In order to solve the problem of plurals, gerundive web form, and in general of suffixes, we use the *Porter Stemmer* [17] to extract the root for every term (both for document words and ontological terms) using these truncated words for the matching.

Moreover, once done the classification the user can see all the documents classified by arguments and ranked by the affinity measure. The user can also submit a query on a particular cluster of documents.

Conceptual classification

Several search engines offer the possibility to classify documents thanks to clustering algorithms which organize in topics the interrelated documents. From our point of view, is the ontology which suggests the classification for the document *corpus*. The user has only to select the discourse domain and specify the depth of sub-clusters in order to avoid a too detailed classification. The sub-cluster hierarchy reflects the structure of the ontology, maintaining the relations of specialization.

By using this policy, the QA asks the OA which reads the ontology (potentially imported by the user) and replies sending the suggested queries, in a hierarchical structure.

The QA then submits the query to the merger agents and, after collecting the results, composes the clusters deleting the possible empty categories.

Query expansion

Query expansion is focused on the user's query. Every word is parsed by the OA in order to suggest alternatives. The user can select three types of integration which can be also applied at the same time. The first one integrates each word which is also included in the selected ontology with specialized concepts. For example, if the user's query is *car retailer* and *car* is an *automobile ontology* concept which is specialized in *station wagon*, *coupe*, and *convertible*, the query is rewritten in this manner: *(station wagon retailer) OR (compact retailer) OR (coupe retailer) OR (convertible retailer)*, allowing the user to access also these pages where the term *car* is not explicitly cited. Another type of integration similarly extends the query to those terms which are related to the query keywords through properties (*owl:ObjectProperty*).

Furthermore, each keyword can be integrated by suggesting possible synonyms specified in the given ontology. For this reason we use the owl constructs *owl:sameAs* and *owl:equivalentClass*. Incidentally, this third type allows, potentially, the multi-language support, if the concepts are translated in several languages.

4.1 Constructing ontologies

A possible objection to the use of ontologies for expanding the abilities of AgentSeeker is that the explicit construction of an ontology is a complex and time consuming task which makes onerous the document retrieval process, in term of work resources involved in the system setup.

Fortunately, the reuse of ontological representations is now relatively simple, if we consider, for example, the fact that *Swoogle* indexes about 10000 online ontologies.

Another scenario which makes the ontological support more profitable is an additional tool we are developing: a sort of Wikipedia for ontologies (a similar project is Ontowiki [18]). The goal is to build an ontology with the help of the components of a social group. For example, being AgentSeeker aimed to the industry, every employee could contribute adding or enriching concepts. In this way, the union of the single competences allows the formal definition of the company's knowledge, helping AgentSeeker to provide results in line with the users' skills.

This social framework for constructing ontologies manages also user's accounts and rights, and maintains a versioning system and a module for changes tracing.

4.2 A simple case study

In order to illustrate the potentialities of AgentSeeker, we deployed a federation of multi-agent platforms to monitor web sites and local documents pertaining to *7th Framework Programme* of the European Union and in general European projects. Starting from a list of four web sites (*cordis.europa.eu*, *ec.europa.eu www.welcomeurope.com*, and *www.esf.org*) and a local repository, we cumulated information on approximately 304000 documents stored in two indexes of about 1 GB. Five computers are involved (an Intel XEON dual core 2 GHz and 1.5 GB of RAM, three AMD Athlon 2 GHz and 960 MB of RAM, and a Intel Pentium 4 2 GHz

with 512 GB of RAM). They host two merger agents which serve ten indexing agents. The Xeon PC hosts a manager, the two mergers, two indexers, a query agent and the ontology one. With a rate of about 6000 pages per hour, in two days we indexed 1000 web sites and a local repository of 100 documents. We have built also an ontology describing the relevant concepts of the *European projects* domain (Fig 2 shows a fragment of this ontology) and some little ontologies describing the research fields we are interested in. In this way, every user can choose its domain in order to refine the research.

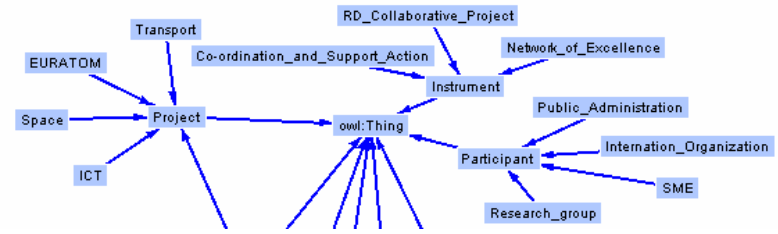

Fig. 2. a fragment of the *European Projects* Ontology

Although an evaluation of this deployment of AgentSeeker is mainly subjective, we report some evidences which illustrate the performances of the platform in an exemplificative way. For example, by using the ontology representing concepts in the domain of European projects and searching for *project call*, the ontology agent classifies the page *http://cordis.europa.eu/fp6/ projects_call.htm* with a high rating of 0,02309, also *http://www.sos112.lt/index/en/front?page=2&/* has a good rating 0,01385 because is a European funded project. *http://www.apache.org/foundation /how-it-works.html* has instead a low rating because does not match with the given ontology: 0,00477. By using this classification, the retrieved documents are listed in order to privilege the most relevant ones, according to the aforementioned rating. From this point of view the end-user is facilitated because the risk to examine a false-positive, consistently decreases. Another demonstration regards the query extension. If the user searches for *project proposal* and *project* is a concept specialized in the typical sectors (*ICT*, *EURATOM*, *Transport*, *Space*, etc.), the query is extended by the Ontology Agent in this way: *(project proposal) OR (ICT proposal) OR (EURATOM proposal) OR…* and we notice an increasing of the found documents equal to 258%. Most of the new entries are enlisted in the first positions, so against the sensible growth of listed pages which could make the research more difficult, the user finds first the potentially relevant documents. Also without *a priori classification* (in case the ontology is dynamically imported by the user and it is not directly supported by AgentSeeker) the performances are encouraging: in the first 60 hits only 7 pages are completely off topic, while with no *query expansion* the erroneous documents are 21.

Considering the achievements presented above, we can states that the result of a single AgentSeeker query often corresponds, if we use a common search engine, to a tedious/time consuming user query process which involves the submission of some queries and the manual integration and ranking process of the obtained results.

The usability of AgentSeeker cannot be significantly proved through objective parameters because its effectiveness is evaluated by the end-user on the basis of its experience and skills. Nonetheless, we can state that in general the users appreciate a domain-oriented classification which helps them to not lose bearings, also against a

growth of the found pages due to the query expansion; a growth which, anyhow, does not affect the quality of results because the unimportant documents are relegated in the last positions by the ranking algorithms (both *a priori classification* and Lucene internal ranking).

5 Conclusions and future works

Although AgentSeeker is a functioning prototype, we consider it an on-going project and a base for further improvements and experiments. The multi-agent system warrants a solid platform on which to develop Semantic Web applications related to the management of large amount of documents. Intrinsic scalability and adaptability of agent-oriented architectures make AgentSeeker able to tune itself to different scenarios and contexts: from a little academic laboratory which wants to manage its collection of papers, to the large enterprise which wants to keep the lid on its document corpus.

We use ontologies in order to formally describe the domains where AgentSeeker is called to operate. Presently, the ontology utilization can be considered basic and subject to further improvements. For example we could develop a behaviour for our ontology agent able to reason about the concepts and their relations, in order to find implicit associations and properties. Moreover, explicit properties are now considered as simple links between two concepts; a future improvement will enable the ontology agent to consider, in some way, the meaning of the property.

We plan to introduce also the possibility to explore the web, indexing only those sites which are relevant considering the ontologies included in the AgentSeeker repository. An indexing agent will visit few pages and then asks the ontology agent to determine if the web site is relevant.

To index coherent documents and pages will allow users to add an upper layer to the application (for example, federating a new platform with new types of agent roles) in order to do market researches, business intelligence processes, learning-from-text techniques, etc.

In conclusion, we think that AgentSeeker contributes to the improvement of search engine performances, combining a multi-agent system with ontological representations. By using Lucene.NET and homemade spiders, AgentSeeker covers the whole process, from the document parsing to the storage of extracted data. This feature assures full control of every aspect, in respect to other solutions which implement meta-search engines leaning on results of online search engines operations. Also the comparison with [6] follows this criterion. In [6] the use of ontologies seems rather fine and elegant. On the other hand, it fully depends on Google for the research of documents and requires that the user formulates a detailed query on which the system constructs the concept network. The solution we propose is then more pragmatic and voted to limit the user's efforts in order to develop a system which could be used in the everyday work (or life) activity.

References

1. Ghemawat, S., Gobioff, H., Leung, Sh.: The Google File System. In: 19th ACM Symposium on Operating Systems Principles, pp. 20--43, New York (2003)
2. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. In Communications of the ACM, vol. 51, no. 1, pp. 107--113 (2008)
3. Koshman, S., Spink, A., Jansen, B.: Using clusters on the vivisimo web search engine. In: HCI International. Lawrence Erlbaum Associates, Mahwah, Las Vegas (2005)
4. Gauch, S. and Wang, G., Gomez, M.: ProFusion: Intelligent Fusion from Multiple, Distributed Search Engines. In: Journal of Universal Computer Science, vol. 2, pp. 637--649, (1996)
5. Hu, M.: MAGI: Multi-AGent-Indexing A Fusion Re-Ranking Meta-Search Engine. Technical Report, University of Waterloo.
6. Kanteev, M., Minakov, I., Rzevski, G., Skobelev, P., Volman, S.: Multi-agent Metasearch Engine Based on Domain Ontology. In: Autonomous Intelligent Systems: Multi-Agents and Data Mining, vol. 4476/2007, pp. 269--274, Springer Berlin / Heidelberg, (2007).
7. Linn, C. N.: A multi-agent system for cooperative document indexing and querying in distributed networked environments. In: International Workshops on Parallel Processing, IEEE Computer Society, Wakamatsu, Japan (1999)
8. Kraines, S. and Guo, W. and Kemper, B. and Nakamura, Y.: EKOSS: A Knowledge-User Centered Approach to Knowledge Sharing, Discovery, and Integration on the Semantic Web. In: Journal of information processing and management, vol. 50, pp 322, Springer, (2007)
9. Esmaili, K.S., Abolhassani, H., Neshati, M., Hariri, B.B.: MOSSE: A Multi Ontological Semantic Search Engine. In: ASWC2006 Workshop on Web Search Technology, Beijing, China (2006)
10. Doulaverakis, C., Nidelkou, E., Gounaris, A., Kompatsiaris, Y.: An Ontology and Content-Based Search Engine for Multimedia Retrieval. In: 10th East-European Conference on Advances in Databases and Information Systems, ADBIS, Thessaloniki (2006)
11. Chiba. E., Ogura. K., Kameyama. W., Nakano, M., Kodo, y., Tsutsui, E.: Asia Broadband Experiment on Ontology-based Search Engine. In: distance learning and the Internet conference, Tokyo (2008)
12. Passadore, A., Incao, G., Pezzuto, G. De Laurentiis, R.: Smart Search: a Tool Supporting Knowledge Extraction and Automatic Classification of Documents. In: WCC08, 20th World Computer Congress 2008, Milano (2008)
13. Vecchiola, C., Grosso, A., Passadore, A., Boccalatte, A.: AgentService: A Framework for Distributed Multi-agent System Development, accepted for publication on International Journal of Computers and Applications, ACTA Press (2009)
14. Vecchiola, C., Grosso, A., Boccalatte, A.: AgentService: a framework to develop distributed multi-agent systems. In: International Journal of Agent-Oriented Software Engineering, vol. 2, no.3 pp. 290 -- 323, (2008)
15. Foundation of Intelligent Physical Agents (FIPA), http://www.fipa.org.
16. Apache Foundation, http://lucene.apache.org.
17. van Rijsbergen, C.J., Robertson, S.E., Porter M.F.: New models in probabilistic information retrieval. British Library, chap. 6, London (1980)
18. Auer, S., Dietzold, S., Riechert, T.: OntoWiki-A Tool for Social, Semantic Collaboration. In: Lecture notes in computer science, vol. 4273, Springer (2006)