Indexing enterprise knowledge bases with AgentSeeker

Andrea Passadore
University of Genova
Via Opera Pia 13, 16145 Genova
+390103532284
passa@dist.unige.it

Alberto Grosso
University of Genova
Via Opera Pia 13, 16145 Genova
+390103532284
agrosso@dist.unige.it

Antonio Boccalatte
University of Genova
Via Opera Pia 13, 16145 Genova
+390103532812
nino@dist.unige.it

ABSTRACT
The aim of this paper is to introduce AgentSeeker: a multi-agent platform for indexing local and online textual files, with the semantic contribution of domain specific ontologies. These ontologies describe the application domain and the competences the user is referring to, during the interaction with the platform, namely a query session.
AgentSeeker is addressed to enterprise applications, thanks to its flexible and scalable structure. Companies with significant electronic knowledge bases can take advantages from AgentSeeker, for their business activities. The value added of AgentSeeker is represented by an *Ontology Agent* which 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 many 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 bound to the user's query. In order to achieve this goal, two aspects of AgentSeeker are 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 amounts 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. THE CURRENT PANORAMA OF SEARCH ENGINES
For most of the Internet surfers, the quintessential search engine is *Google*. Surely, it represents the largest and most ambitious project of indexing electronic data. For this reason we introduce Google, in order to cite the common issues in developing and maintaining search engines.

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 documents, and e-mails. It stores them in a local index which is constantly updated.

An intermediate perspective between the personal hard disk and the whole Internet, is represented by 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 2.1) which represents a significant improvement in the search engines panorama. Other noteworthy tools are *Northern Light (www.northernlight.com)*, which includes also a sort of document classification based on taxonomies; *Search Engine Studio (www.xtreeme.com/search-engine-studio/)*; *Noematics Reflexion (www.noematics.com)*; 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.

Oftentimes, computer users clash with the efficient but sometimes not very intelligent actions 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 (www.iboogie.com)*, *SnakeT (snaket.di.unipi.it)*, *KWMAP (www.kwmap.net)*, and *Exalead (www.exalead.com)*.

2.1 Agent-based search engines and the ontological support

At now, solutions involving agents or ontologies are essentially proposed by the research community, with no contributions from the industrial world, confirming that their potential is not yet completely expressed.

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 project, 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

AgentSeeker has been developed by following few basic principles: *scalability*, *flexibility*, and accurate 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. Multi-agent systems ensure the requested flexibility and scalability. Moreover, crawlers which explore the network are often described as autonomous software agents. Our goal is then to build an agent-oriented architecture which supports crawler agents parsing documents on the net.

In order to consider documents not only as alphanumeric sequences 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 corpora, 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 these practical reasons AgentSeeker can only manage the textual content. Ontologies are used to describe the knowledge of the user, his competences, and

his expectations in order to apply them during the document search.

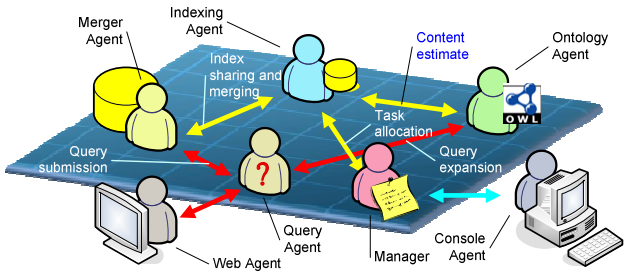

Figure 1: The agent roles presently implemented in AgentSeeker.

3.1 The AgentService framework

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]; in [13] there is also a comparison with other mature and well-known multi-agent frameworks) 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 instantiated 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.

3.2.1 The Indexing Agent

The *Indexing Agent* (IA) is the core of the system. It is the agent role in charge of downloading documents from web sites or from local storages (both distributed on an intranet and stored in a single PC). The IA supports text extraction on usual *html* files, simple text documents (txt), all the formats of the *Microsoft Office Suite* (in particular *Word*, *Excel*, *PowerPoint*), and also *pdf* files. Parsing hypertexts, the agent extracts also the hyperlinks and distinguishes from *internal links* (namely pages which belong 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 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. Since a task could last for tens of minutes or entire hours, this (de)registration process is not so critical for the yellow pages service. Also in the case the manager agent (see 3.2.7) is searching for a free indexing agent and all the IAs are busy, the manager agent does not submit continual requests to the yellow pages service, but makes the queries less frequent in order to avoid an useless overcharge of the service (cyclically the frequency of queries is reset). This technique ensures a strong flexibility of the system. In this way, we can add further instances of indexing agents and the merger agent must not know in advance the name of every IA. Furthermore, it is not the manager agent which maintains the list of free indexing agents: following this policy, we open the platform to possible expansions, where other manager agents or other new agent roles can assign tasks to free IAs, simply querying the yellow page service.

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 the various session indexes coming from IA instances are merged.

3.2.2 The Merger Agent

Every IA manages its own local index where it stores the text extracted in the current session. In order to avoid the proliferation of too much local indexes on which the user has to submit a query, the *Merger Agent* role (MA) has been created in order to collect the results of the IAs once they have finished their indexing sessions. The MA manages a central index (based on Lucene) where the user's queries are materially executed. For this reason the MA has two main tasks: to maintain a central repository of the indexed texts and to respond to the queries coming from the query agent. These two tasks are implemented as two concurrent behaviours: the first is in charge of receiving a link to a remote partial index representing the result of an IA indexing session. This behaviour extracts each record from the remote partial index and inserts it into the central index, removing possible duplicated entries and optimizing the repository, compressing the textual data. The second behaviour is devoted to the submission of queries to the central index through the Lucene API; once the results have been provided by the Lucene API, the MA sends them to the query agent.

The role of the MA becomes essential in case we deploy a federation of distributed AgentSeeker platforms (see 3.3). In this case, a set of IAs can be associated to a specific MA ensuring that all the information extracted by each IA is neatly stored in a centralized facility. While IAs can be ephemeral entities which could be increased or decreased in number, the MAs represent fixed reference points for the other AgentSeeker agent roles: especially the query agent.

3.2.3 The Query Agent

The *Query Agent* (QA) receives a textual query from the outside. Possible senders could be the *Web Interface Agent* or the *Administration Console Agent*, two external agents which directly interact with a user. The QA is able to reply to whatever agent interested in submitting a query to AgentSeeker. The essence of the QA is to furnish a standard querying interface easily useable by third-part applications integrated or linked to our MAS. In brief, the QA hides the complex AgentSeeker topology (in particular the existence of merger agents and distributed indexes).

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.

3.2.4 The Ontology Agent: an agent which holds the enterprise's knowledge

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 (http://razor.occams.info/code/semweb/)* and *Linq to RDF (http://www.hookedonlinq.com/LINQTORDF.ashx)*. 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.

The reason of the OA existence is mainly the need of separating the semantic core of the application from the content extraction and management. Presently, the OA is a starting point which will be subject to further improvements, for example introducing behaviours which implement reasoning techniques. Other behaviours will be able to interact with external knowledge bases, as WordNet or Swoogle from which they will extract information useful to integrate user's queries.

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 (for further details, see 4.1.1).

3.2.5 A web interface agent to interact with users

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).

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

3.2.7 The manager agent: 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 instantiated. 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 instantiated 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 transparent 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. REPRESENTING KNOWLEDGE IN AGENTSEEKER

The performances of the AgentSeeker agent community are improved through the use of ontological representations.

As introduced before, ontologies help AgentSeeker agents by modeling the discourse domain the user is taking into account when he submits a query. Concepts and relations constituting the given discourse are represented by RDF – OWL files.

Based on OWL ontologies, AgentSeeker, in its first functioning prototype provides three policies which exploit the explicit semantic representation of the enterprise's knowledge.

4.1.1 A priori classification

A first strategy 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 the URIs are included 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 the classification has been completed, 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.

4.1.2 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, it 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.

The conceptual classification is the most complex policy which requires time and resources, because several queries must be sent to the MA (one for each concept). Lasting up to tens of seconds (depending on the ontology size), the conceptual classification could be cyclically applied by the system and results presented to the user without re-submitting the whole query set.

4.1.3 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.

Considering the Case study presented in 4.3 we noticed that performances are encouraging, with responses provided within three or four seconds.

4.2 Building ontologies with social collaboration

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 the document retrieval process onerous, 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.

The interactive construction and maintenance of the ontologies supported by AgentSeeker introduces the problem of updating the *a priori* classification results on the basis of the modifications occurred in the ontologies. Periodically, AgentSeeker indexes again the document corpus in order to detect possible changes in the textual contents. During this phase the indexing agent takes into account modifications in the ontologies in order to update the estimate of arguments, running from scratch the classification algorithm.

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

4.3 A 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 the *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* 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.

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 state 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 his 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

At now, AgentSeeker is a fully working prototype, subject to several improvements in term of usability and performances. Moreover, it represents a platform on which we can build specific applications that require large textual repositories to process. Integrating a new application in AgentSeeker is a relatively simple process, because it is necessary only to add a platform (or just agents to the federation) and interact with the usual AgentSeeker agent roles. As possible scenarios we identify market researches, business intelligence processes, learning-from-text techniques, etc. Besides the flexibility ensured by agent-oriented architectures, we can exploit also their intrinsic scalability and adaptability making AgentSeeker able to tune itself to different 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 ask the ontology agent to determine if the web site is relevant.

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 as storage facility 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.

6. REFERENCES

[1] Ghemawat, S., Gobioff, H., Leung, Sh. 2003. The Google File System. In: 19th ACM Symposium on Operating Systems Principles, pp. 20--43, New York.

[2] Dean, J., Ghemawat, S. 2008. MapReduce: Simplified Data Processing on Large Clusters. In Communications of the ACM, vol. 51, no. 1, pp. 107—113.

[3] Koshman, S., Spink, A., Jansen, B. 2005. Using clusters on the vivisimo web search engine. In: HCI International. Lawrence Erlbaum Associates, Mahwah, Las Vegas.

[4] Gauch, S. and Wang, G., Gomez, M. 1996. ProFusion: Intelligent Fusion from Multiple, Distributed Search Engines. In: Journal of Universal Computer Science, vol. 2, pp. 637—649.

[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. 2007. Multi-agent Meta-search Engine Based on Domain Ontology. In: Autonomous Intelligent Systems: Multi-Agents and Data Mining, vol. 4476/2007, pp. 269--274, Springer Berlin / Heidelberg.

[7] Linn, C. N. 2009. 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.

[8] Kraines, S. and Guo, W. and Kemper, B. and Nakamura, Y. 2007. 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.

[9] Esmaili, K.S., Abolhassani, H., Neshati, M., Hariri, B.B. 2006. MOSSE: A Multi Ontological Semantic Search Engine. In: ASWC2006 Workshop on Web Search Technology, Beijing, China.

[10] Doulaverakis, C., Nidelkou, E., Gounaris, A., Kompatsiaris, Y. 2006. An Ontology and Content-Based Search Engine for Multimedia Retrieval. In: 10th East-European Conference on Advances in Databases and Information Systems, ADBIS, Thessaloniki.

[11] Chiba. E., Ogura. K., Kameyama. W., Nakano, M., Kodo, y., Tsutsui, E. 2008. Asia Broadband Experiment on Ontology-based Search Engine. In: distance learning and the Internet conference, Tokyo.

[12] Passadore, A., Incao, G., Pezzuto, G., De Laurentiis, R. 2008. Smart Search: a Tool Supporting Knowledge Extraction and Automatic Classification of Documents. In: WCC08, 20th World Computer Congress 2008, Milano.

[13] Vecchiola, C., Grosso, A., Passadore, A., Boccalatte, A. 2009. AgentService: A Framework for Distributed Multi-agent System Development, accepted for publication on International Journal of Computers and Applications, ACTA Press.

[14] Vecchiola, C., Grosso, A., Boccalatte, A. 2008. AgentService: a framework to develop distributed multi-agent systems. In: International Journal of Agent-Oriented Software Engineering, vol. 2, no.3 pp. 290 – 323.

[15] Foundation of Intelligent Physical Agents (FIPA), http://www.fipa.org.

[16] Hatcher, E., Gospodnetić, O., McCandlessvan, M. 2009. Lucene in action. Manning Publications Co, Greenwich.

[17] Rijsbergen, C.J., Robertson, S.E., Porter M.F. 1980. New models in probabilistic information retrieval. British Library, chap. 6, London.

[18] Auer, S., Dietzold, S., Riechert, T. 2006. OntoWiki-A Tool for Social, Semantic Collaboration. In: Lecture notes in computer science, vol. 4273, Springer.