AgentSeeker: an ontology-based enterprise search engine

Extended abstract

A. Passadore, A. Grosso, and A. Boccalatte, *University of Genova, Via Opera Pia 13, 16145 Genova*

Abstract— **The aim of this paper is to introduce AgentSeeker: a distributed 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. They are used by an Ontology Agent which organizes the results of a user's query, according to the concepts which represent the relevant entities in the company business.**

AgentSeeker is addressed to enterprise applications, thanks to its flexible and scalable structure managed by the Federation Management Suite, which ensures a comfortable administration of the distributed platform, balancing the computational load.

Index Terms— **Document, multi-agent system, ontology, search engine, MAS federation.**

I. Introduction

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, thanks to the *AgentService Federation Management Suite*.

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. A particular kind of agent is able to manage ontologies, integrating the user's queries with semantically related words, discovered through the analysis of concept 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. And exactly the management of this distributed environment is a relevant topic of this extended abstract and it represents an improvement in respect with the first version of AgentSeeker presented in [1].

II. AgentSeeker

AgentSeeker has been developed by following few basic principles: scalability, flexibility, and accurate management of textual documents. 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.

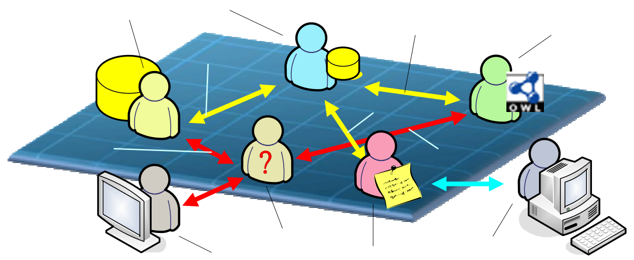

Figure 1: The agent roles presently implemented in AgentSeeker.

AgentSeeker is essentially based on *AgentService* [2], a framework for the development and execution of multi-agent systems, implemented in the C# programming language and by using the Microsoft .NET libraries. An overview of the features and the programming paradigm provided with

AgentService are detailed in [2] and [3].

A. *Designing the AgentSeeker overall system*

Following the agent oriented paradigm, AgentSeeker is designed as a society of interoperating agents. All the involved components are modeled as agents playing a specific role within the community. Figure 1 shows the different roles and their interactions. In addition to the usual AgentService agents, equipped with *behaviours* and *knowledge objects* (here referenced as *internal agents*), AgentSeeker includes also *external agents*: namely external applications which act like ordinary agents for easily interacting with the rest of the platform. Finally, in order to make agents act in a coordinated way, a set of interaction protocols are defined for modeling their interoperations.

The Indexing Agent

The *Indexing Agent* (IA) is the core of the system. The main goal of the IA is to index documents and inform the *Merger* and the *Manager Agent* about the completion of its work. The documents to be indexed are collected from web sites or from local storages, both distributed on an intranet and stored in a single PC. The IA is able to extract text from common *html* files, simple text documents (*txt*), *pdf* files, and all the formats of the *Microsoft Office Suite*. 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).

IA promotes its services by registering to the yellow pages service included into the system. When IA receives a new job, it first deregisters itself from the yellow pages till the job will be completed, then it renews the registration becoming available again for executing a new job. For each indexing session, IA 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* [5]. Essentially developed to store textual contents and to operate queries on them, Lucene is a scalable solution that allows the implementation of large architectures.

The session index is then accomplished in collaboration with the *Merger Agent,* as shown in the next paragraph.

The Merger Agent

Once the IAs have created and stored their partial indexes, there is the need to merge them in order to speed up the search operation by avoiding fragmentations. The *Merger Agent* role (MA) has been defined in order to collect the results of the IAs once they have finished their indexing session. 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.

Considering the relation between the dimension of the index and the search speed, *AgentSeeker* provides the possibility to configure the system for involving more than one MA.

The Query Agent

The *Query Agent* (QA) manages the search requests coming from the users through the *Web Interface Agent* and the *Administration Console Agent* or from different external applications. Once a request is received, the QA checks if the ontological support has been requested; in this case, it contacts the Ontology Agent in order to have a semantic support for improving the results. Then, it sends a request to all the MAs and collects the consequent results. Finally, it furnishes the results to the web-based interface agent.

Mastering the enterprise's knowledge through the Ontology Agent

The Ontology Agent (OA) is the keeper of the knowledge of the system. Its functionalities will be fully described in the section IV 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. In particular, this service is used by the IA during its indexing sessions, which then receives an estimate of the arguments dealt in the examined text.

The Manager Agent

The *Manager Agent* plays the role of orchestrator, coordinating the activities of the other agents. It is in charge of distributing and balancing the workload among the agents and it acts as supervisor monitoring the index and search processes. 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 workload is then naturally balanced on every machine and every agent.

The agent-based web interface

From the user's point of view, AgentSeeker is a simple web application with a look-and-feel similar to the usual search engines. In the back-end of the web application, an agent is in charge of interacting with a remote AgentSeeker installation in order to submit the queries. The interoperation between AgentSeeker and the web application is based on a web service interface exposed by *AgentService*. 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. From the web page, the user can select or import an ontology which represents the argument he has in mind during his query session.

The Administration console

Similar to the previous one, another 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 shut 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.

III. The Federation Management Suite

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 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.

A. Managing a distributed environment

As shown before, it is plausible a complex federation of platforms with a common goal. In this case, AgentSeeker becomes a distributed society of agents which must act and react in unison. From the point of view of the administrator, it is very important to manage the whole system in a centralized way. Although the pseudo-agent playing the role of *Administration Console* represents a useful tool for tuning some parameters of AgentSeeker and monitoring the activity of the running agents, it is however an entity of the platform with no power on the life-cycle of peers or even of the entire platform.

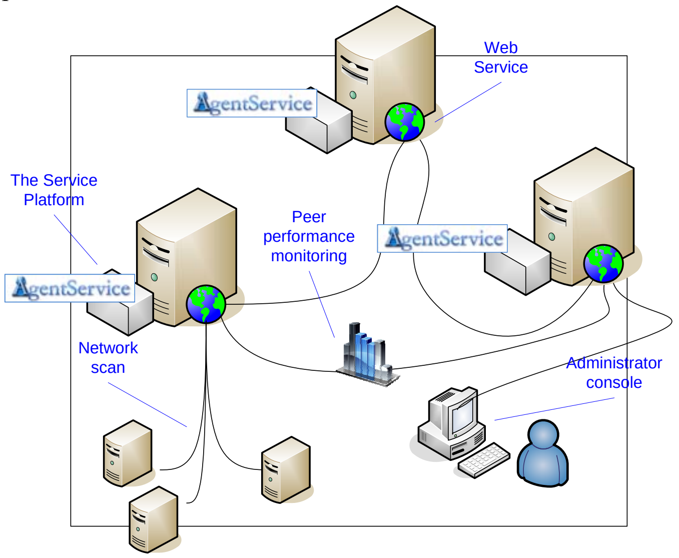

Figure 2: The management infrastructure of a federation.

For this reason we developed an infrastructure which allows a system administrator to easily manage distributed AgentService applications, both at the level of platforms and of agents. AgentSeeker represents a significant test-bed thanks to its intrinsic distribution of resources and scalability.

This infrastructure is a sort of cloud for MASs which enables the administrator to:

1) Install a platform on a PC
2) Join the platform to a particular federation of platforms
3) Discover new platform nodes
4) Deploy an application on one or more distributed platforms
5) Manage the platform life-cycle
6) Create, execute, move, and stop single agent instances.

B. A platform as a service

Fig 2 shows the topology of a federation of AgentService platforms. Each platform is, first of all, a service running on a computer. Installed as a *Windows Service*, each platform is instantiated at the computer wake up. The platform service exposes a *web service* through which an administrator can set the state of the platform: *idle*, when it is not yet launched; *ready*, when every platform module is loaded; *running*, when the platform is ready to execute agents; *stopping*; and *shuttingdown*.

Every platform-service runs a thread which is in charge of discovering new nodes on the same subnet. Every IP is scanned in order to check if a possible peer platform is listening. In this case, a handshake procedure starts in order to determine if the candidate is available for joining the federation (on the same subnet several federations can coexist and every platform is set up to join one, more, or all the federations). Periodically every node polls the federated platforms to check their existence and to share the list of discovered peers. In this way, the federation is updated, thanks to the interactions of the federated nodes. From the practical point of view, the list of federated platforms is used by the AgentService messaging module to physically route the messages to remote agents.

C. *The management of a federation*

Once the platforms are physically installed on a network, the administrator can manage the federations connecting its administration GUI to the web service exposed by whichever remote platform. This platform represents the access point of the federation and all the commands coming from the administration GUI transit through it. From the GUI, the administrator is able to see the node list and the status of the related platforms. Further information, constantly updated, regard the CPU occupancy, the available RAM, and, if the platform is running, the number of hosted agents.

The administrator can upload to a single node, or automatically to the whole federation, a set of files like assemblies containing the templates of agents, configuration files, etc. In this manner, a new multi-agent application can be rapidly deployed and executed.

D. *Add an agent to the federation*

During the usual execution of a distributed multi-agent application as AgentSeeker, it is possible to add new agent instances in order to increase the available resources in the federation. Moreover, it could happen that an agent crashes for some reason and must be stopped and replaced. In these cases, the administrator must be able to create new instances of agents. A first possibility is to select a particular node and then to launch a new instance. Another way is to select the whole federation and create an agent instance suggesting no physical destination. Every node of the federation knows the status of each peer node and is able to find the platform with more free resource. In order to do it, the node classifies the peer performances (CPU time, free RAM, and number of agents) evaluated on a time slot (typically 10 minutes). Once the best platform is selected, the current node checks if the destination has the necessary assemblies containing the agent template and related classes and only if the platform candidate satisfies all the requirements, the agent is materially instantiated on that node.

E. *Moving agents instances*

A federation of platforms is an environment where agent activities and interactions evolve in time. An agent could suddenly increase its needs of resources or densely interact with a remote agent. In these cases, the presence of the agent in the current platform could deteriorate the node performances or saturate the network link with a high throughput of messages. As described in [6] the AgentService team developed a facility for moving agents from a platform to another one, saving their state and then resuming them on the destination. The agent state is represented by its collection of knowledge objects which are persisted and sent to the destination along with the needed assemblies.

Considering the GUI we are describing, the administrator is able to select an agent and, evaluating the performances of the other nodes, he can stop it, select the destination, move the agent and resume its activities on the new node, balancing the computational load of the federation.

IV. REPRESENTING KNOWLEDGE IN AGENTSEEKER

Ontological models of the discourse domains which AgentSeeker deals with, allow a sensible growth of the multi-agent application performances.

The aim is to help the user during the submission of a query, taking into account the argument he is considering in order to automatically add more details to the interrogation submitted into the system.

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

A. *A priori classification*

This strategy is applied during the indexing sessions and requires a strict interaction between the IA which extracts the text and the OA which classifies it with the help of its knowledge represented by ontologies.

The idea is to estimate the affinity of each processed text with the topics modeled in the ontologies directly supported by AgentSeeker. The MAS maintains a repository where it stores the core ontologies. Since AgentSeeker is able to manage the OWL language, it can accept further ontologies imported by the users. For this reason we can state that AgentSeeker is not tuned on a particular set of ontologies, but it is up to deal with every ontology.

From the practical point of view, every document stored in the Lucene index has a particular field where the URIs of the supported ontologies are related to an estimate of its relevance in respect with the supported semantic models. The measure is expressed in term of percentage of document words which are also described in the ontology. We apply the Porter Stemmer algorithm [7] in order to extract the root of each term suppressing any suffix (plurals, gerundive forms, etc.) and we remove the so-called *stop words* (adverbs, conjunctions, etc.)

from the document, because of their irrelevance.

Once the classification has been completed, the user can order the results on the basis of their relevance according to the arguments supported by AgentSeeker. Usual queries and selection of documents by arguments can be integrated.

B. *Conceptual classification*

In AgentSeeker we can exploit ontologies in order to classify documents on the basis of a natural hierarchy suggested by the relations of specialization and generalization among concepts. The user suggests the depth of the sub-cluster hierarchy in order to avoid a too detailed classification.

Following this policy, the QA obtains from the OA a hierarchical structure whose nodes represents both the concepts and the single queries to submit to the MA and then to Lucene.

Furthermore, it is possible to restrict the number of processed documents, by clustering only the results of a usual query.

C. *Query expansion*

The third strategy is aimed to expand the user's query. The OA parses the query in order to add alternatives or more details. According to the argument the user is considering (namely the ontology), the QA analyzes each query term in order to check if it is also an ontological concept. In this case the term can be expanded by following up to three types of policies. The first policy integrates the query, adding, for each word which occurs also in the ontology, all the 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.

V. 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 to interact with the usual AgentSeeker agent roles. The Federation Management Suite ensures a comfortable tool for managing large deployments of platforms, relieving the administrator from any effort for balancing the computational load. 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 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. The solution we propose is then more pragmatic and voted to limit the user's and the administrator's efforts in order to deploy a system which could be used in the everyday work (or life) activity.

REFERENCES

[1] A. Passaodore, A. Grosso, A. Boccalatte, "Indexing enterprise knowledge bases with AgentSeeker", WOA 2009, From Objects to Agents, Parma, Italy, July 2009.

[2] C. Vecchiola, A. Grosso, A. Passadore, and A. Boccalatte, "AgentService: A Framework for Distributed Multi-agent System Development," to be published in *International Journal of Computers and Applications*, ACTA Press, 2009.

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

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

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

[6] A. Boccalatte, A. Grosso, and C. Vecchiola, "Implementing a Mobile Agent Infrastructure on the .NET Framework," in *Proc. 4th International Conference in Central Europe on .NET Technologies*, Plzen, Czech Republic, May, 2006.

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