SEMANTIC WEB BASED PROACTIVE SEARCH FOR ENTERPRISE

Li Li, Feng Liu and Wu Chou
Avaya Inc. 233 Mt. Airy Road, Basking Ridge, NJ 07920, U.S.A.

Keywords: Semantic web, REST web service, Proactive search, Text annotation, Software agent, Implicit social network, Expert finder.

Abstract: This paper presents an approach and a software architecture based on agent and web service technologies to support proactive search to enrich enterprise communication and collaboration. In particular, we combine software agents and REST web services to deliver relevant information found from RDF databases to the users without interrupting their workflows. The relevant information includes text annotations, implicit social networks, and recommended experts. We discuss how service composition can be used to efficiently combine results from distributed functions to support independent and scalable semantic web development. Initial experimental results indicate the proposed approach is feasible and efficient.

1 INTRODUCTION

Enterprise communication and collaboration typically involves email, meeting schedule, voice, video, chat, wiki, blogs, and various forms of documents, such as design documents and bug reports related to products and service offerings. As different organizations adopt technologies in different paces, these rich sets of digital content often exist in different formats managed by different systems that are often not connected to each other. For example, emails between group members are stored in special format at local disks or email servers. The product design documents are managed by some special proprietary relational database, whereas the project progress is tracked on a separate group wiki site. As a consequence, a lot of valuable information is buried in disparate computers that are not readily accessible.

As the number of media types and the amount of digital contents increase, it can cost significant overhead and a reduction in productivity, as users may have to spend extra effort searching for the relevant information. For instance, when scheduling a project planning meeting between several groups, people typically receive invitations in email that have a subject, time and location, a short description, and some attachments and links. If people need to find out more background information about the participants, the previous history of contacts on this subject, or new products related to this project, people have to do searches using several special applications. As every participant repeats almost identical searches to become informed, the productivity of the enterprise is reduced as the number of participants increase.

To address this problem, we present a knowledge agent based approach for enterprise, derived from two related technologies - semantic web and proactive search.

The semantic web in our approach refers to a set of technologies based on the web architecture and knowledge representation languages including RDF (RDF) and OWL (OWL 2004). The semantic web technologies offer a set of solutions to address the heterogeneous data problem in enterprises, whereas URI provides a uniform identification mechanism for data in enterprises, and RDF provides a uniform representation language about the relations between those identified data. In addition, HTTP provides a uniform protocol to access the distributed data, and SPARQL provides a declarative way to query those data. Moreover, ontologies offer a way to integrate RDF graphs from different sources. Because semantic web technologies are based on Description Logic (RDF 2004), they also offer a framework to reason and inference about the data.

Despite these advantages, a challenge to adopt semantic web for enterprise is how to transform the raw enterprise data into RDF. It is ideal but

Li L., Liu F. and Chou W..
SEMANTIC WEB BASED PROACTIVE SEARCH FOR ENTERPRISE.
DOI: 10.5220/0003474806490656
In *Proceedings of the 7th International Conference on Web Information Systems and Technologies* (SWAT-2011), pages 649-656
ISBN: 978-989-8425-51-5

unrealistic to force all enterprise systems and applications to expose their data according to a predefined ontology. Instead, we need to allow organizations to evolve their semantic web incrementally and independently. To support this path, we adopt REST (Fielding 2000, Richardson 2007) web service paradigm as our semantic web infrastructure, because REST is optimized for such distributed hypertext systems. Unlike conventional approaches to semantic web that aim to support linking and querying of raw RDF triples (Linked Data), we focus on developing knowledge based web services that can enhance enterprise communications. In particular, we investigate how to develop a scalable and robust REST architecture that can share and compose distributed knowledge web services across organizations.

Proactive search pushes relevant information to users without user's asking for it specifically. It is a departure from current interactive search paradigm in several aspects. In interactive searches, a user composes a specific query, enter it into a search box, select results and integrate them into his application manually. Albeit being quite flexible, interactive search has some disadvantages and limitations. Firstly, the interactive search mode usually forces a user to leave his current activity and work on a separate search activity. Secondly, the query does not carry the context from which the search is launched. Thirdly, the results of interactive search depend on the quality and accuracy of user's query. Fourthly, to integrate the search result back into the user's workflow and context, it typically requires user's manual operation.

In proactive search, a user's communication activity is treated as the query to the search engine, thereby providing the necessary context for more accurate results. Instead of asking a user to select the results, proactive search integrates relevant information directly into the communication activity in a nonintrusive way. As a result, the user can focus on his business activities without taking detours to seek for relevant information. For example, an incoming email or an outgoing email can be treated as query to the proactive search engine. The relevant information found about the topic, people or products mentioned in the emails is integrated into the emails as hyperlinks. The disadvantage of proactive search is that the input is limited to current user's activity and context. The second challenge in adopting semantic web technologies is how to determine what is relevant given a context as this is significantly more complex than most queries. In open domain search, these are very difficult problems. However, as enterprises have more organized and predictable activities and workflows than individual users, we can use those patterns in enterprise data to help tackle these tough problems.

Proactive search can be supported by client-server architecture as interactive search. However, the clients in proactive search assume more responsibility than in interactive search. In proactive search, the clients are software agents that monitor user's activities and invoke the corresponding knowledge base web services to obtain the right information at the right time.

The rest of the paper is organized as follows. Section 2 presents the overall architecture of our semantic web based system. Section 3 briefly discusses the knowledge transformation process. Section 4 presents some functions and services built into our approach and architecture. Section 5 discusses the agents and applications based on the described functions and services. Section 6 is dedicated to implementation and experimental results. Section 7 reviews some related work, and we conclude this paper with Section 8.

2 OVERALL ARCHITECTURE

To support semantic web based proactive search, we need to provide customized semantic web based functions that are targeted to different business environments. For example, in a call center, we need functions that classify emails, annotate important concepts in emails, and suggest relevant responses. In a group collaboration application, we need functions that bring up contact history on a subject and show common interest between participants. However, due to the limitation of SPARQL, many of these functions cannot be implemented as SPARQL queries to RDF databases. For this reason, we decide to expose these functions as REST services that are sharable and reusable across organizations. REST services encourage distributed and independent development of services, which is one of our design goals. Besides connecting different applications, our REST composition approach allows us to distribute a semantic function that is too large for one machine to multiple machines in parallel, and use service composition to aggregate the distributed logic.

On the client side, our software agents are embedded in user's communication and collaboration applications. These agents monitor user's activities, retrieve relevant information from the REST services, and inject the relevant information into the collaboration environment.

Figure 1 illustrates the high level components in our REST composition architecture.

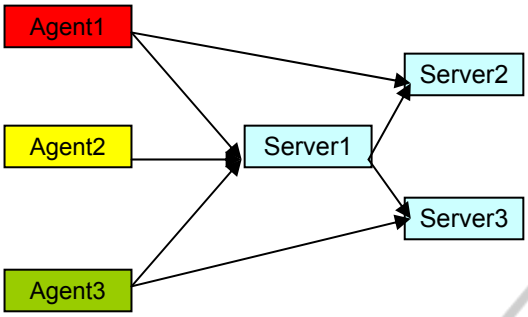

Figure 1: High level components of REST architecture.

By driving the agent states with hypertext from the REST services, this REST architecture offers the following degrees of freedom for adopting independent changes. At the server side, a new service can be deployed without having to reconfigure all the agents. An existing service can be upgraded without breaking those agents that use the service. On the client side, agents can acquire different "skills" required for different environments by following the hyperlinks to different REST services. Although in this architecture, the agents do not need to directly communicate with each other, they can still collaborate indirectly by sharing their states through the servers. Agents and servers can also use content negotiation to find the best representation for a given situation.

Each server in our architecture builds the REST services from the knowledge base in layers as depicted in Figure 2.

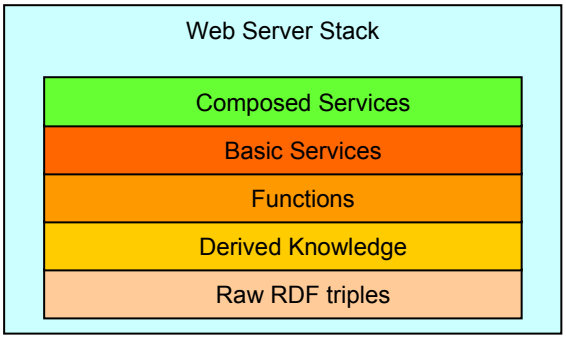

Figure 2: Web Server Stack.

The bottom layer is the raw RDF triples collected from various sources of enterprise data, including relational databases and web pages. On top of this, it is a layer of the knowledge derived from the RDF triples - some of those triples are collected off-line, and some of them are derived dynamically based on queries. A function takes an input query and produces some outputs based on the derived knowledge. The basic REST services expose these functions as resources, and the composed REST services invoke the basic or composed services at local and remote servers to carry out a task.

Service composition is a process that implements a service by combining outputs from other services. This process can be used to break a large semantic database into small ones and distribute a related function into a set of servers that form a tree structure. The servers in the leaf nodes offer the basic services, while the servers in the interior nodes offer partially composed services. As the result, the server at the root node offers the completely composed services. Because the services are stateless, a composed service invokes its children services in parallel and merges the results for its parent. This process is illustrated in, Figure 3 where "local" means some local services involved in the compositions.

Notice that this composition architecture is different from the conventional computer cluster architecture which has a fixed entry point and a specified topology. Instead, in our case, each distributed function may have a different entry point and topology of its own. For example, Figure 3 illustrates two distributed functions with entry points Server 1 and Server 3 respectively. Server 1 is the entry point to the server tree consisting of Servers 1, 2, 4, 6 and Server 3 is the entry point to the server tree consisting of Servers 2, 3, 4, 5.

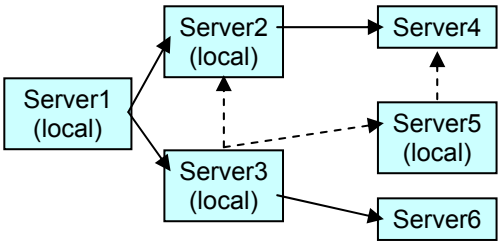

Figure 3: Service compositions of two distributed functions with solid and dotted lines respectively.

3 RDF TRANSFORMATION

As enterprise data often exist in different forms and formats, they have to be transformed into semantic web first. This transformation contains two steps. First the data are transformed into web resources that have unique URI. Second the metadata are extracted and transformed into RDF triples, often with the help of public and private ontologies. For

structured data, such as relational databases, this transformation is straightforward as outlined in (RDB2RDF). In our study, we have transformed a relational database about documents with 160,699 records into 3,182,721 triples following the Dublin Core (Dublin Core) ontology. The subject is the document URI in all triples. The following are some sample triples with sensitive information replaced by generic strings:

```
<uri_1>
<http://purl.org/dc/elements/1.1/title>
"Title 1" .
<uri_1>
<http://purl.org/dc/elements/1.1/creato
r> "Author 1" .
```

Our experience shows some problems for well-defined data. First, it turns out that many important relations (predicates) are not in Dublin Core or other ontologies that we know of. We created and added private ontologies to cover them, but they cannot interoperate easily outside this domain. Second, many data fields, such as author names, are not properly entered in the database. Many names have variations that make matching and cross-reference from other RDF databases difficult.

4 FUNCTIONS AND SERVICES

Within our proposed REST architecture, we develop several functions and services for semantic web based proactive searches.

4.1 Entity Annotation

Entity annotation function takes an incoming text and produces a set of annotations for the entities in the text based on current knowledge. An annotation is a 4-tuple `(phrase, start, length, link)` that identifies the phrase being annotated, the starting position and the length (both in characters), and the concept related to the phrase. When clicked, the link will open a web page showing the detail information.

To support this function, we first index the RDF triples on selected predicates. This dramatically reduces the indexing and search space from entire literals to the selected literals. Another technique to save memory and improve efficiency is to avoid creating separate index. Instead, we pre-process RDF triples by tokenizing the literals into phrases so that they would match the tokenized input. The outcome of this process is a many-to-many mapping from indexed phrases to concepts.

The annotation algorithm is a modification of the left-to-right maximum tokenization algorithm (Guo 1997) developed for Chinese language processing. The algorithm aims to find the longest token sequence from left to right that matches an indexed phrase in the RDF triples and record the corresponding concept. If more than one concept is found, the server creates a link representing a list of matches. Unlike the traditional tokenization algorithm that covers the entire text, our algorithm skips unmatched tokens. The high-level components of this function are illustrated in Figure 4.

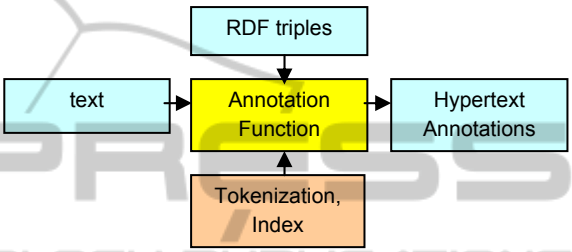

Figure 4: Components of the annotation function.

This function is exposed as a REST service on a designated resource. There are two ways to invoke the service: HTTP GET and HTTP POST. GET is used for short text, and POST is used for long text. The REST service returns related annotations in the following formats: 1) JSON for agents in web browsers; 2) HTML for direct rendering in web browsers; 3) HTML tables for embedding and debugging in web browsers; and 4) Prolog terms for efficient service composition.

When this function is distributed on a tree of servers (Figure 3), the parent server sends a copy of the text to all its children in parallel, which will return the annotation results (or faults). Once the parent server receives all the results (within a timeout interval), it merges what it received so far into a coherent annotation and sends it back to its parent or the client if it is the root server. To maintain the longest phrase condition, the merge process removes all covered phrases. In other words, if a server returns an annotation for phrase `x` and another server returns a phrase that contains `x` as a substring, then the first annotation is removed from the merged annotation. The merged annotation therefore will contain only longest phrases.

4.2 Implicit Social Network

In enterprise, people communicate and collaborate on daily basis. These activities form a social network

that is dynamic and implicit with rich relations associated with social contents, e.g. email, IM, co-authorship, etc. This implicit social network can be discovered by inspecting the artefacts of these activities, such as email exchanges and authorships of project documents, etc. Because there are very rich relations between people in an organization, our semantic web approach is well suited for representing and discovering such implicit social networks. In our study, we find the following relationships being important for a person in implicit social networks (Table 1).

Table 1: Relations in implicit social network.

Relation	Comment
collaborators	Collaborators of this person
followers	People interested in this person
citations	Artefacts that cite this person
products	Artefacts made by this person
expertise	Expertise of this person

To cope with the dynamic nature of the implicit social network and to save memory, these relations are derived by rules in response to incoming queries. The input of this function is a URI identifying a person, and the output is the relations about the persons found in the knowledge base. Because this information is to be consumed by human users in our system, the current output supports only HTML in which relations are represented as hypertext. The high-level components of this function are illustrated in Figure 5.

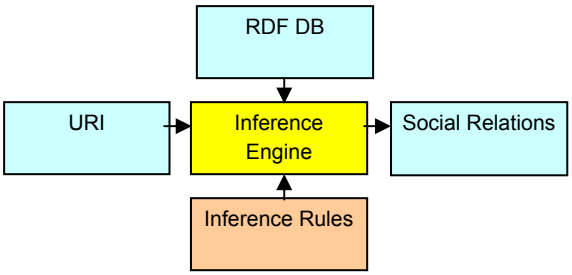

Figure 5: Implicit Social Network function.

When calculating these relations, a relation with more recent and frequent activities are valued more than one with infrequent activities in the past, as people in the organizations can take different roles over time. The activities in a relation are therefore weighted by an exponential decay function to reflect this time-based relevance value. Assume that a relation R is derived from a set of n activities each with timestamp T_i, then the time value of this relation, with respect to the current timestamp T, is calculated as follows, where C is a normalizing factor and λ is a scaling factor:

$$tv(T,\lambda,C,R) = C\sum_{i=1}^{n} e^{-\lambda(T-T_i)},\ R=\{T_i \mid 1 \le i \le n\}$$

Similar to the annotation function, when this function is distributed, the parent server sends a copy of the URI to its children servers in parallel, which return the relations (or faults). The parent server then merges the results using set unions.

4.3 Expert Finder

In many enterprise systems, there is a need to find experts with certain skills, so that a problem can be directed to the most qualified persons. In an organizational environment, since we value team work and influence as much as individual skills, we need to find experts who not only have required expertise, but also have high reputation and authority.

The expertise of a person can be evaluated based on the products he produces or contributes. For example, if a person designed several web servers, it is reasonable to assume he is an expert in that area. In current system, the expertise of a person is represented as a vector where each dimension corresponds to a skill and the value indicates the strength of the skill. To speed up the matching process, a training process is used to compute and save the expertise vectors for each person in the knowledge base. For a person p with an expertise vector e, the relevance of p with respect to a given problem description vector x can be calculated using the vector space based semantic model as the angle between x and e:

$$relevance(p) = cos(x, e)$$

The reputation of a person can be evaluated based on how other people evaluate his work. This is calculated with bounded recursion and loop detection based on the implicit social network as follows, where *rating(p)* reflects the total evaluation a person received:

$$reputation(p) = rating(p)+\sum reputation(followers(p))$$

The authority of a person captures how much power a person has in an organization. It can also be calculated with bounded recursion and loop detection based on implicit social network as follows, where *level(p)* corresponds to the power level a person has in an organizational hierarchy:

$$authority(p)=level(p)+\sum authority(collaborators(p))$$

The input to the expert finder function is a text describing a problem, and the output is a ranked list

of experts. The input text is first converted to a vector to search for persons whose expertise is above a threshold. The candidates in the list are then re-ranked by averaging the normalized relevance, reputation and authority scores. The high-level components of this function are illustrated in Figure 6.

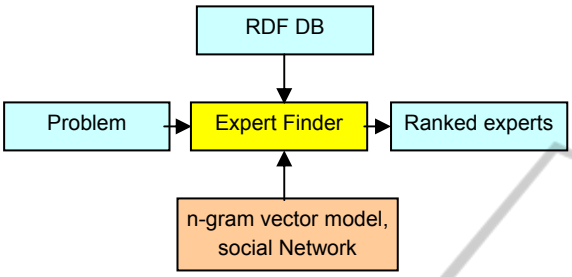

Figure 6: Expert finder function.

When this function is distributed, a parent server sums up the scores returned from its children servers for the same person. The returned experts are then merged and re-ranked to pass back up to the parent or agent.

5 SOFTWARE AGENTS AND APPLICATIONS

The proposed architecture, functions, and services were used and applied to several enterprise communication and collaboration systems. All the applications use the same REST services, but they differ in how the agents behave.

5.1 Browser Agent

The browser agent in our study is a Firefox extension that monitors and annotates the web page a user is viewing. A user can activate and deactivate the agent from the browser menus and ask the agent to annotate the current page or restore the original page. The user can also configure the agent to use a different REST service by entering a different URI. Figure 7 is a screenshot of the interface to our browser agent in front of the Firefox browser window.

5.2 Call Center Agent

This software agent in our study assists human agents in call centers by finding relevant information in incoming contacts (e.g. emails) to save human agents from searching for them. Each incoming email was intercepted by the software agent to find and identify the important concepts and recommend experts related to the email using our REST services. The email was then enriched to embed found information as hyperlinks into the original email, and forwarded to the system. When a human agent receives this enriched email, he can click the links to obtain the detailed information.

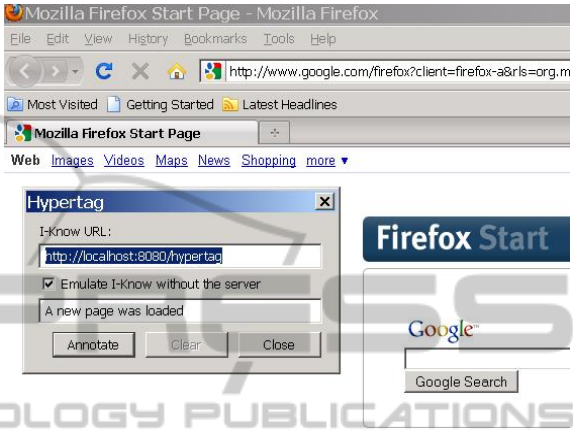

Figure 7: A software agent embedded in Firefox browser.

Figure 8 is a screenshot of an annotated email in our prototype Call Center system as seen by a human agent. For privacy, any personal identification information is whitened out.

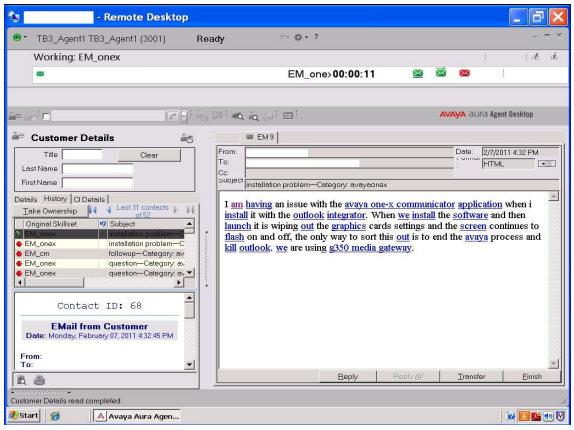

Figure 8: An annotated email in call center client.

5.3 Google Wave Agent

We developed a special software agent for Google Wave using Google Robot API (Google Wave) to monitor and annotate multi-party group chat in real-time. To enable this agent function, a user just needs to invite our software agent to the current Wave conversation. When a user clicks a button to finish

his chat, the agent will invoke the REST services and annotate the chat text with hyperlinks that point to the relevant concepts and information to bridge the semantic gap in collaboration. These hyperlink annotations are propagated to all participants of the Wave session in near real-time. Any user can click a link in the chat window to visit it.

6 IMPLEMENTATION AND EXPERIMENTS

We implemented a prototype REST server system using SWI-Prolog (SWI-Prolog). The HTTP servers were Prolog images (executable program) compiled for different machines. Our current RDF database contains triples from three sources as listed in the following table (Table 2).

Table 2: Size of knowledge sources.

Source	Triples
Wikipedia (2006/03/26)	47,054,407
database	3,182,721
product	75
total	**50,237,203**

To simulate distributed REST services that contain different knowledge bases, the Wikipedia RDF file was broken up at random into 10 small sets each with up to 5 million triples. This resulted in 12 text files in N-Triples format. These 12 files are loaded into SWI-Prolog, indexed and converted into binary SWI-Prolog RDF database format for efficient loading. The following table (Table 3) compares the size (KB) of text files with the size (KB) of binary databases. This table shows that for most large files, the binary format has an over 80% size reduction.

Table 3: Comparison of size reductions.

Name	Text	Binary	Ratio
WP_0	753243	148159	19.67%
WP_1	753268	149143	19.80%
WP_2	753986	147896	19.62%
WP_3	753976	147493	19.56%
WP_4	754396	149513	19.82%
WP_5	753430	148539	19.72%
WP_6	753364	148594	19.72%
WP_7	753890	149200	19.79%
WP_8	753979	149268	19.80%
WP_9	309799	67885	21.91%
database	380004	73624	19.37%
product	10	7	70.00%

These binary databases were loaded into the memory of different machines according to their capacity. This distributed configuration allows us to recruit different number of non-dedicated machines, ranging from powerful servers to even notebook computers. The smallest system that provides satisfactory performance for the entire 50+ million triples consisted of two Linux machines (3.0 GHz CPU/4 GB RAM and 1.6 GHz CPU/4 GB RAM), each with about 25+ million triples.

To test the performance of the distributed servers, we selected 3 Wikipedia binary databases WP_{0,1,2}, and distributed them into three server trees in a LAN environment. The first server tree had one root node containing all 15 million triples; the second server tree had one root with 5 million and one child with 10 million triples; the third tree had one root and two children, each with 5 million triples. In all these trees, the root server was a Windows 2003 Server machine with Dual Core (3.0 GHz and 2.99 GHz) and 2GM RAM and the child servers consisted of two Linux machines mentioned above. To test the performance of these trees, a test text of 1142 characters was sent 10 times to the root server which returns 30 annotations. The average service execution time was recorded using Prolog `time/1` predicate on the root server and summarized in the following table (Table 4) with standard deviations. The execution time includes time for local function, service composition as well as logging.

Table 4: Performance of three server trees.

Server tree	Avg. Time (second)
1 node	0.406 (0.0003)
2 nodes	0.390 (0.0179)
3 nodes	0.401 (0.0321)

Our results showed that a distributed function may outperform its local version when it is distributed over faster machines. When the two Linux machines with more RAM were used, the average service execution time on the root was improved slightly (2 and 3 vs. 1). Also the test showed that parallel distribution of a function to two nodes created only a small overhead compared to distribution to one node (3 vs. 2).

7 RELATED WORK

There have been active researches in how to process large scale RDF databases (Cai 2004, Urbani 2009, Ianni 2009, Husain 2009, Large Triple Stores). But

the focus of these efforts is different from ours. First, their focus is limited to efficient storage and retrieval of large datasets whereas ours is to support general computing and inference over the datasets. Second, the approaches have been based on a homogeneous architecture where a set of computers either use a single protocol or form a fixed topology, whereas our REST composition service based approach does not assume any single protocol or topology.

There has been some work on using RDF database to annotate text (Schönhofen 2008, Ferragina 2010). But these systems are special cases of proactive search that we propose. In addition, they do not propose a general architecture to support distributed functions.

Our REST service architecture is also different from the conventional 3-tier web architecture consisting of data, business logic and presentation. In our architecture, the presentation is not consumed by end users but by agents. Unlike business logic that accesses local data, our logic can access distributed functions through service composition.

8 CONCLUSIONS

The contributions of this paper are summarized below:

- We proposed a software architecture by combing software agents and REST web services to support distributed and scalable semantic web development;
- We demonstrated that this architecture can effectively support proactive search to enrich enterprise communication and collaborations;
- We demonstrated that service composition is a feasible approach to efficiently combine distributed functions;
- We implemented several agents in different use cases and a prototype system with 50+ million RDF triples;

The future work will be focused on collecting more RDF data and develop more advanced algorithms, functions and services.

ACKNOWLEDGEMENTS

We would like to thank Mr. Jan Wielemaker for answering our technical questions about SWI-Prolog. We also thank Mr. Jack Barnard for providing access to a large document database.

REFERENCES

RDF. Resource Description Framework (RDF), http://www.w3.org/RDF/, last accessed, 10-Feb-11.

OWL 2004. *OWL Web Ontology Language Overview*, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/owl-features/, last accessed 10-Feb-11.

RDF 2004. RDF Semantics, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/rdf-mt/, last accessed 10-Feb-11.

Fielding, Roy T., Architectural Styles and the Design of Network-Based Software Architectures, Ph.D. Dissertation, 2000, http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm, last accessed 10-Feb-11.

Richardson, L.; Ruby S., Restful Web Services, O'Reilly, 2007.

Linked Data, http://linkeddata.org/, last accessed 10-Feb-11.

RDB2RDF, Use Cases and Requirements for Mapping Relational Databases to RDF, W3C Working Draft, 8 June 2010, http://www.w3.org/TR/rdb2rdf-ucr/, last accessed 10-Feb-11.

Dublin Core, Dublin Core Metadata Initiative, http://dublincore.org/, last accessed 10-Feb-11.

Guo, J., Longest Tokenization, Computational Linguistics and Chinese Language Processing, Vol. 2, No. 2, August 1997, pp 25-46.

Google Wave, http://wave.google.com/about.html, last accessed 10-Feb-11.

SWI-Prolog, http://www.swi-prolog.org/, last accessed 10-Feb-11.

Cai, M.; Frank, M., RDFPeers: A Scalable Distributed RDF Repository based on A Structured Peer-to-Peer Network. WWW 2004 Proceedings of the 13th International Conference on WWW, pp 650-657.

Urbani, J. et al, Scalable Distributed Reasoning using MapReduce, ISWC 2009, Vo. 5823, pp 634-649.

Ianni, G. et al, Dynamic Querying of Mass-Storage RDF Data with Rule-Based Entailment Regimes, ISWC 2009, pp 310-327.

Husain, M. F. et al, Storage and Retrieval of Large RDF Graph Using Hadoop and MapReduce, Lecture Notes in Computer Science, 2009, Vol. 5931, pp 680-686.

Large Triple Stores: http://www.w3.org/wiki/LargeTripleStores, last accessed 10-Feb-11.

Schönhofen, P., Annotating documents by Wikipedia concepts, 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology.

Ferragina, P.; Scaiella, U., TAGME: On-the-fly Annotation of Short Text Fragments (by Wikipedia Entitities), CIKM' 10 Proceedings of the 19th ACM international conference on Information and knowledge management, pp 1625-1628.