# Auto-complete for Improving Reliability on Semantic Web Service Framework

Hanmin Jung[1], Mi-Kyoung Lee[1], Won-Kyung Sung[2], and Beom-Jong You[1]

[1] Information Technology Research Lab., KISTI, Korea
[2] Dept. of Policy Research, KISTI, Korea
{jhm,jerryis,wksung,ybj}@kisti.re.kr

**Abstract.** This paper presents two methods for enhancing auto-complete which providing search keywords that the user wants. The first is to display only search keywords that can guarantee a successful search result in real time regardless of document's insertion, deletion, and update. The second is to display search keywords with their entity types such as person, institution, and topic. To accomplish them, we introduce an auto-complete table that stores the entities extracted and indexed from input documents and their document frequency (DF). An auto-complete manager checks whether each entity in the table can guarantee a successful search result or not by considering its DF, and provides proper entities with their types to the user. To verify the effect of the auto-complete, we are designing a comparative experiment. OntoFrame 2007 without the functions will be compared with OntoFrame 2008 with the functions for discovering the effect of our auto-complete on the reliability of Semantic Web services.

**Keywords:** Auto-complete, Semantic Web, Semantic Web Framework, OntoFrame, Reliability, Document Indexing.

## 1 Introduction

Auto-complete is a feature for predicting a word or phrase that the user wants to type in without the user actually typing it completely. Auto-complete on the Web is usually implemented by Ajax (Asynchronous JavaScript and XML) which is one of key technologies classified in Web 2.0 (see fig. 1). They are widely applied over the Web sites including digital libraries, commercial portals, and enterprise applications [1]. It is expected to be applied more and more in the viewpoint of enhancement of the user experience. However, most auto-completes simply display search keywords retrieved from the user's query logs and system dictionaries without considering the quality of search results. The reliability of such auto-completes would be dropped in case that the user selects a search keyword which can not generate a successful search result. It is not a special case for small/medium enterprise portals because they suffer from relative short of contents compared with commercial portals such as Google, Yahoo, and Amazon. Even worse, different types of search keywords are mixed in their auto-complete lists. It compels the users to look up whole of the list for finding a

M.J. Smith and G. Salvendy (Eds.): Human Interface, Part II, HCII 2009, LNCS 5618, pp. 36–44, 2009.

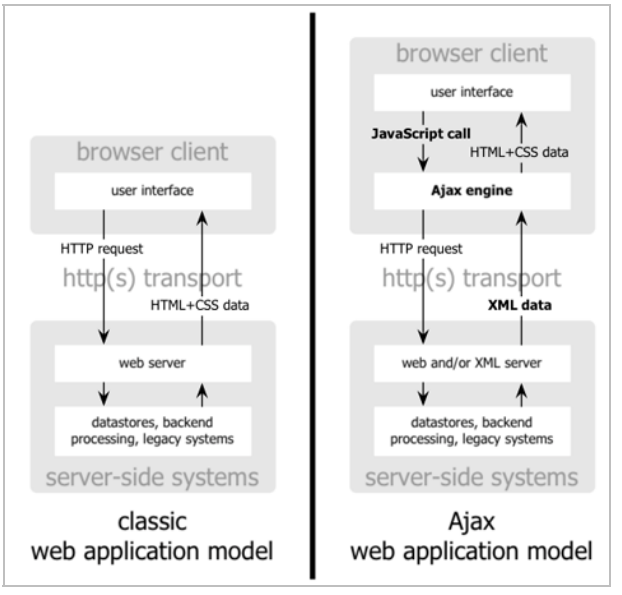


**Fig. 1.** Classic Web application model and Ajax Web application model[1]

search keyword they want. These problems can be solved by introducing two methods that display only search keywords that can guarantee a successful search result and provide their entity types such as person, institution, and topic. This research deals with the way how the two methods can be achieved on a Semantic Web service framework.

## 2 Related Studies

Han and Lee deals with costs and benefits of internet search with costumers' prospective [2]. They reveal the fact that pains during the search are closely related with the user satisfaction, thus it should be designed to minimize the time and effort required. Auto-complete is an important function in that it can reduce the pains of the user by providing a way to select a search keyword without typing it completely. Another study on the usability of auto-complete concludes that the user satisfaction and search efficiency (the time needed to complete a given task) are affected by the function [3]. It also reports many positive comments were received from the users. Even for mobile devices, auto-complete enables the users to finish their tasks with fewer errors [4]. The study has clearly proven its advantages in terms of satisfaction, efficiency, and stability.

However, studies on auto-complete enhancement are comparatively insufficient against its importance. A patent created by Lee and Wales proposes auto-complete using multiple dictionaries in ways of lookup and merge [5]. Auto-complete introduced

[1] http://www.adaptivepath.com/images/publications/essays/ajax-fig1_small.png

by Miki et al. can be classified as an advanced function in that it deals with ontology and data conversion [6]. An application form is generated in ways of referring the ontology data that knowledge managers have constructed manually. When the users input data in the form, the auto-complete function recognizes language types and field types in order to convert the data into appropriate values. However, the study is not closely concerned with ours because we concentrate on only the construction of search keywords in auto-complete list. A study on auto-complete for predicting Chinese characters with partial uses a prefix tree decoder [7]. It also adopts speech recognition for supporting bi-modal Chinese character input. Liu et al. found the bi-modality improves the input speed, but their research scope does not match with us exactly.

Bangalore et al., as a significant study on auto-complete for improving reliability, introduces a method for providing search keywords that are able to generate a successful search result in the UMLKSK interface [8]. It uses a flag to mark the success or failure of search results. In case of success, the corresponding search keyword is marked with the flag. This method ensures a successful search result by displaying only the search keywords by considering the flag when the auto-complete list is provided. However, the auto-complete list would be occasionally out-of-date because the marking occurs only when the user enters a search keyword. Even after adding an input document which includes a search keyword that has caused the failure previously, the keyword will not be display in the auto-complete list until the user types manually the search keyword without referring to the list.

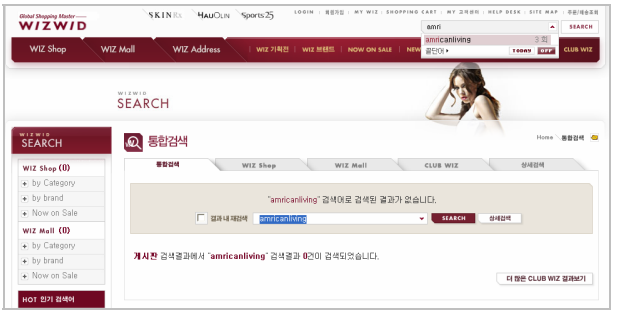


**Fig. 2.** Example of a search result with failure caused by selecting an improper search keyword in auto-complete list ('Wizwid' online shopping mall)

Many popular Web sites, even including enterprise portals, provide auto-complete for helping the users to find search keywords in ease. They usually use the popularity of the users' input regardless of success/failure of search results as shown in fig 2. Because they do not consider incremental data add-up, which would cause mismatch problem between auto-complete and search results, search using auto-complete occasionally fails. As a new technology, Google offers keyword suggestions in real time in the form of auto-complete. However, mismatch in the number of documents occurs as shown in fig. 3. It may be caused by incomplete incremental indexing

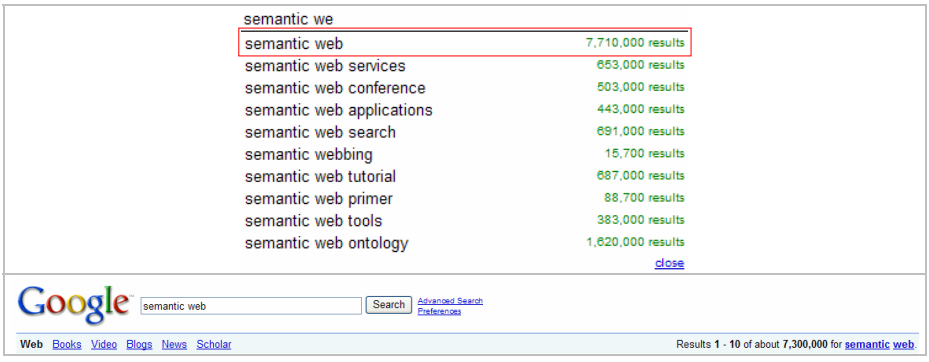


**Fig. 3.** Example of mismatch between auto-complete and search results in the number of documents ('Google' suggest; 7,710,000 results in auto-complete and 7,300,000 in search results)

management related with auto-complete. The following section explains how we resolve these problems with an auto-complete manager and an auto-complete table.

## 3 Auto-Complete in OntoFrame

OntoFrame, a Semantic Web service framework [9] [10], aims at search and discovery of science and technology information using Semantic Web technologies. It includes a search engine and a reasoning engine. The former searches full-text documents and the latter discovers implicit knowledge by exploiting relations between instances, i.e. entities. OntoFrame gathers legacy data and transforms them into RDF (Resource Description Framework) triples by referring to predefined ontology schema designed for a specific application domain. The reasoning engine expands the triples at idle time, i.e. forward chaining, using user-defined rules, and puts back the results into an RDF triple store. OntoFrame-based service communicates with the two engines through Web Services, SPARQL (Simple Protocol and RDF Query Language) queries, and XML documents. OntoFrame provides entity-centric unified search[2]. Predefined entities are managed in a URI server which is a semantic data management tool with the RDF triple store. The server is referred by a document indexer for acquiring entities and their types from an input document (see fig. 4).

Auto-complete is applied to OntoFrame for helping the user to search with convenience and efficiency. To sustain reliability, auto-complete should provide search keywords that guarantee successful search results. However, the previous version of OntoFrame (OntoFrame 2007) provided a simple auto-complete function which just shows a search keyword list matched with the user's input string. The keywords are pre-extracted and refined topic keywords from test collection. Even the case that a keyword never appears in indexed documents, it can be displayed in auto-complete list,

[2] Entity-centric unified search can be defined as a unified search generating a Web page which consists of service components selected dynamically according to the user's input corresponding with an entity reserved in the system. Different kinds of search result pages would be generated in case of different entity types.

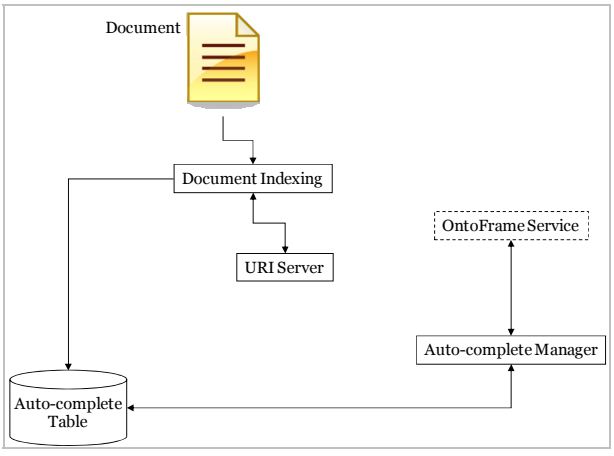


**Fig. 4.** Auto-complete process

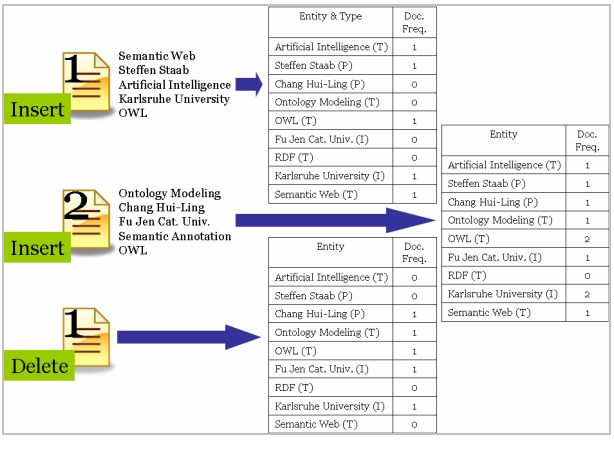


**Fig. 5.** Example of changes in the auto-complete table according to document's indexing (P: person, I: institution, T: topic)

which will cause a failed search result. Even worse, the use of all of the topic keywords increased the load of auto-complete. We also found the users' reliability of the service is degraded when they select improper keywords, i.e. keywords that can not guarantee a successful search result, in the list. Thus, this version of OntoFrame (OntoFrame 2008) introduces an auto-complete manager for guaranteeing successful search results from selecting a search keyword in auto-complete and further increasing usability by displaying entity types additionally. After extracting entities from an input document, the auto-complete manager updates the document frequency (DF) of the entities in an auto-complete table, i.e. DF of the entities would increase by one as shown in fig. 5.

When the user enters a search keyword, the auto-complete manager finds the entities matched with the keyword by looking up the auto-complete table. Then, it checks whether DF of each retrieved entity is zero or not. Those entities with more than zero are displayed in the auto-complete list. Since the entities in the list are extracted from indexed documents, search results generated from them would be always successful. This method is different from that of Bangalore et al. because DF of entities is updated instantly according to document's indexing, and only proper entities are always displayed in the auto-complete list [8]. The following pseudo codes explain how the auto-complete manager deals with the auto-complete table for generating an auto-complete list corresponding with the user's input.

```
manage_auto_complete(String) {
  Entity[] = get_matched_entity_list(String);
  Valid_entity[] = check_doc_frequency(Entity[]);
  sort_entity_by_type(Valid_entity[]);
  return Valid_entity[];
}

sort_entity_by_type(Entity[]) {
  find_entity_type(Entity[], Person_entity[],
    Institution_entity[], Topic_entity[]);
  sort_entity(Person_entity[]);
  sort_entity(Institution_entity[]);
  sort_entity(Topic_entity[]);
  sorted_entity[] = merge_entity(Person_entity[],
    Institution_entity[], Topic_entity[]);
  return Sorted_entity[];
}

update_auto_complete_table(String[], Operation) {
  While (Entity = get_next_string(String[])) {
    Switch (Opreation) {
      Case Insert: increase_doc_frequency(Entity);
      Case Delete: decrease_doc_frequency(Entity);
    }
  }
}
```

The above functions lookup and manage the auto-complete table. Manage_auto_complete() gets the user's input string from OntoFrame service. It finds the entities of which name is matched with the input string. Check_doc_frequency() checks each entity whether it has DF of more than zero. Entities with document frequency of zero are excluded. By referring the auto-complete table, sort_entity_by_type() makes a merged entity list sorted by their types, e.g. topic and person. Then manage_auto_complete() returns them to the service. DF of each entity is managed by update_auto_complete_table(). The function increases or decreases DF according to operation types, i.e. insert and delete. In case of document's update, insert process would follow delete process. Finally, the auto-complete manager returns a proper entity list as shown in the following example and fig. 6.

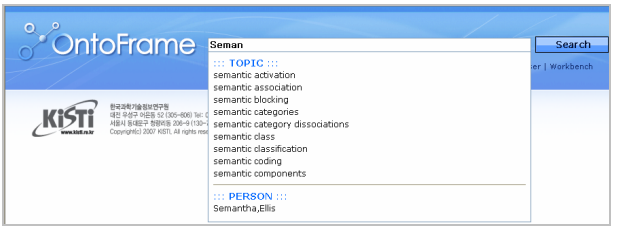


**Fig. 6.** Example of auto-complete in OntoFrame (It shows topic and person entities matched with "Seman.")

API:
EntityList manage_auto_complete(String keyword)
Calling example:
manage_auto_complete("sem")
Result example (see fig. 3):
[semantic activation, Topic]
[semantic association, Topic]
[semantic blocking, Topic]
…
[Semantha,Ellis, Person]

Current number of the entities stored in the service is 923,449 (652,507 for topic and 270,942 for person). We found that 362,319 topic entities (55.53% when compared

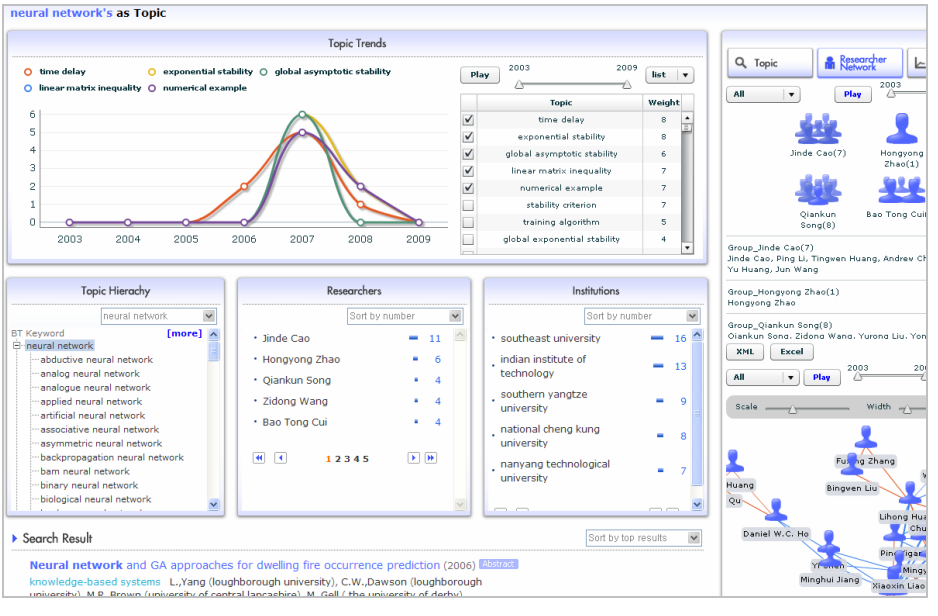


**Fig. 7.** Example of search result generated from topic entity "neural network"


**Fig. 8.** Example of search result generated from person entity "Jinde Cao"

with total topic entities) acquired from indexing of about 200,000 journal papers on information technology and bioinformatics lead to successful search results. Fig. 7 and fig. 8 show examples of successful search results for topic and person entities.

## 4 Conclusion

We introduced an enhanced auto-complete with two functions; displaying only search keywords that can guarantee a successful search result in real time regardless of document's insertion, deletion, and update and displaying search keywords with their types such as person, institution, and topic. For accomplishing them, an auto-complete manager and an auto-complete table are used. The manager gets the user's input string and returns a proper entity list by looking up the table. To verify the effect of the auto-complete, we are designing a comparative experiment. The task to be achieved is "Find a representative researcher and an institution for top 5 topics in the auto-complete list corresponding with 'neural'." OntoFrame 2007 [3] without the functions will be compared with OntoFrame 2008[4] with the functions. We expect to find the effect of our auto-complete on the reliability of Semantic Web services.

## References

1. Beauheim, C., Wymore, F., Nitzberg, M., Zachariah, Z., Jin, H., Skene, J., Ball, C., Sherlock, G.: OntologyWidget – a Reusable, Embeddable Widget for Easily Locating Ontology Terms. J. BMC Bioinformatics 8, 338 (2007)

[3] http://isrl.kisti.re.kr/wsearch/search/main.jsp
[4] http://150.183.113.186:8080/OntoFrame_ISRL/2008_new/main.jsp

2. Han, D., Lee, E.: Exploring the Costs and Benefits of Internet Search from the Online Customers' Perspective: Implications for the Consumer Adoption of the Semantic Web-Based Search Engines. J. Business Education Research 11(1) (2007) (in Korean with English Abstract)
3. Kluge, J., Kargl, F., Weber, M.: The Effects of the Ajax Technology on Web Application Usability. In: International Conference on Web Information Systems and Technologies (WEBIST 2007) (2007)
4. Udyaver, S.: Experimental Comparison of Usability of Hybrid Mobile Devices. In: The 20th Computer Science Seminar (2004)
5. Lee, K., Wales, K.: Methods and Systems for Implementing Auto-complete in a Web Page. US 2004/0039988 A1 (US. Patent) (2004)
6. Miki, T., Ogawa, H., Matsuda, N., Miura, H., Taki, H., Hori, S., Abe, N.: Auto Complete Method for Web Application from Based on Term Hierarchy. In: The 20th Annual Conference of the Japanese Society for Artificial Intelligence (2006) (in Japanese with English Abstract)
7. Liu, P., Ma, L., Soong, F.: Prefix Tree Based Auto-Completion for Convenient Bi-modal Chinese Character Input. In: IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2008) (2008)
8. Bangalore, A., Browne, A., Divita, G.: UMLSKS SUGGEST: An Auto-complete Feature for the UMLSKS Interface Using AJAX. In: AMIA 2006 Annual Symposium (2006)
9. Jung, H., Lee, M., Kang, I., Lee, S., Sung, W.: Finding Topic-Centric Identified Experts Based on Full Text Analysis. In: The 2nd International Expert Finder Workshop at ISWC 2007 + ASWC 2007 (2007)
10. Sung, W., Jung, H., Kim, P., Kang, I., Lee, S., Lee, M., Park, D., Hahn, S.: A Semantic Portal for Researchers Using OntoFrame. In: The 6th International Semantic Web Conference and the 2nd Asian Semantic Web Conference (ISWC 2007 + ASWC 2007) (2007)