RDF Snippets for Semantic Web Search Engines

Xi Bai[1,2], Renaud Delbru[1], and Giovanni Tummarello[1]

[1] Digital Enterprise Research Institute,
National University of Ireland, Galway, Ireland
[2] College of Computer Science and Technology,
Jilin University, Changchun, China
{xi.bai, renaud.delbru, giovanni.tummarello}@deri.org

Abstract. There has been interest in ranking the resources and generating corresponding expressive descriptions from the Semantic Web recently. This paper proposes an approach for automatically generating snippets from RDF documents and assisting users in better understanding the content of RDF documents return by Semantic Web search engines. A heuristic method for discovering topics, based on the occurrences of RDF nodes and the URIs of original RDF documents, is presented and experimented in this paper. In order to make the snippets more understandable, two strategies are proposed and used for ranking the topic-related statements and the query-related statements respectively. Finally, the conclusion is drawn based on the discussion about the performances of our topic discovery and the whole snippet generation approaches on a test dataset provided by Sindice.

1 Introduction

With the development of the Semantic Web, more and more Resource Description Framework (RDF) documents are published on the Web. Therefore, how to find out the most helpful RDF documents effectively and efficiently becomes one of the hottest problems in the Semantic Web community. Due to the structure of RDF documents and their uncontrolled growth, it is hard for non-technician users to understand or decipher the information coded in RDF syntax. So when users search the RDF documents with some specific topics, the returned results (i.e., RDF documents links returned by SWSE [7] or by the previous version of Sindice [8]) are inconvenient for users to recognize which ones they really want. Therefore, expressive resource descriptions should be generated, which will give users the clues and assist them in recognizing the content of the searched results. Some work [1][2][3][4][5][6] have contributed to the verbalization of domain ontologies using natural language processing (NLP), clustering techniques, or analysis of URLs uing language model, but little work has been done to discover the topics and generate snippets from RDF documents. RDF statements ranking is also very important within the process of generating snippets, but it still faces difficulties and has not been further researched nowadays.

In this paper, we propose an approach for summarizing and ranking the content of RDF documents. We use an heuristic method based on the occurrences of

the RDF nodes and the original URLs to discover the topic nodes. We also give a bench of ranking strategies for improving the snippets and provide users with the relationships between their queries and the searched results. Our approach dose not involve any NLP techniques and it is mainly based on the RDF graph structure instead of the tags defined by people. Therefore, it is domain independent and can be used to process the documents written in other Web resource description languages (e.g., OWL) with few modifications. Based on our work, users without any domain knowledge can easily get the snippets from the RDF documents. Meanwhile, based on the data set provided by Sindice, our experimental results indicate that the generated snippets can assist users in better understanding the RDF documents returned by Semantic Web search engines.

The remainder of the paper is organized as follows. Section 2 outlines our framework for generating snippets from RDF documents. Section 3 describes the RDF documents preprocessing before the commence of the generation. Section 4 describes a heuristic method for discovering the topic nodes. Section 5 describes our topic-related-statements ranking algorithm and query-related-statements ranking algorithm. Section 6 describes how to generate the final descriptions for RDF documents. Section 7 describes the template-based process of generating personalized snippets. Section 8 gives the use case of the snippet generation and the performance of our topic node discovery method is compared with the existing methods. The performance of the whole snippet generation process is also evaluated in this section. Section 9 briefly describes the recent related work. Section 10 draws the conclusions and gives our future research directions.

2 Framework for Generating Snippets from RDF Documents

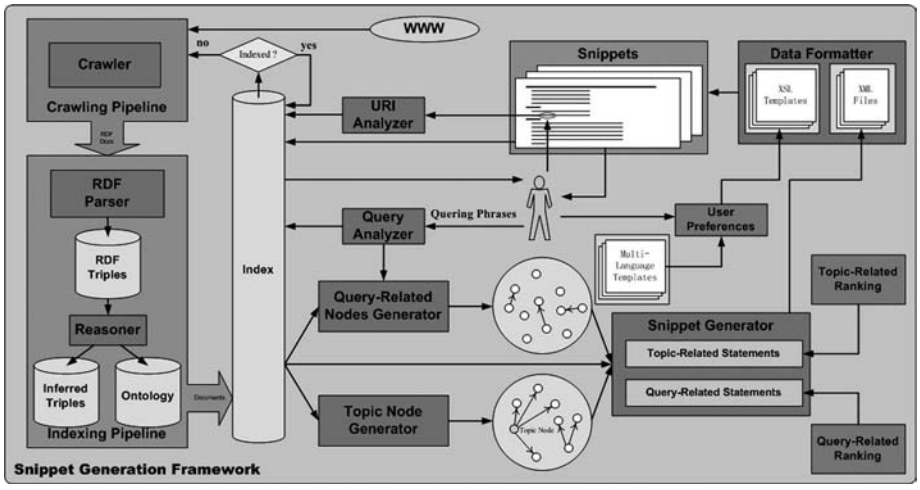


Fig. 1. Framework for Generating Snippets

We describe our approach for generating snippets from RDF documents in this section. The corresponding framework is depicted in Figure 1. In this figure, *Crawler* is in charge of crawling RDF documents from the Web. *RDF Parser* parses each document into a set of RDF statements, which are then saved into the triple repository. *Reasoner* is in charge of using the original RDF documents and their imported ontologies to obtain extra RDF triples. Then the original triples, the inferred triples and the triples in the imported ontologies will be indexed. *Query-Related Nodes Generator* and *Topic Node Generator* take the charge of using heuristic methods to find out the query-related nodes and the topic node respectively. *Snippet Generator* takes the RDF nodes as the input and based on our two ranking algorithms, pre-generates the summarized content which is then saved into the XML documents. According to the users' preferences and predefined XSL templates, *Data Formatter* finally transforms the XML documents into the snippets as the summarizations returned to the users. If users want to get the detail information about the resources contained in the generated snippets, the *URI Analyzer* is capable of acquiring their corresponding URIs of the resources appointed by users and then query *Index* again to recursively generate snippets. If the required resources have not been indexed, *Crawler* will be reactivated to crawl the missed resources. According to the framework, the finally generated snippet is related to both the URL of the original RDF document and the user's query. Therefore, we use the cache to optimize the generation process.

3 Preprocessing of RDF documents

An RDF document contains the resources whose types are possibly not declared in this document. Users will be puzzled, if the majority of the resources appears without their types in the final snippet. Therefore, we first supplement the content of original RDF documents by adding the content of imported ontologies. We also involve the inferring to find out the Inverse Functional Property (IFP) which will assist us in looking for the topic of an RDF document. In [9], the reasoner based on OWL "ter Horst" [10] is used for finding out IFPs. Here, we also use this reasoner to fulfill the inferring tasks. As shown in Figure 1, after the retrieving and the inferring processes, the original RDF document, the ontologies it used and the inferred information will be indexed in N-Triple format.

4 Topic Generation

The RDF documents can be retrieved from the Web in various ways and our RDF dataset is established using Sindice, a scalable online service, which crawls the documents from the Semantic Web and indexes the resources encountered in each source [8]. Before the snippet generation, *RDF Parser* parses them into RDF statements and generates the RDF graphs. Then the subject and the object in each statement are recognized as RDF nodes and each two nodes are connected by a property. Our first step for generating snippets is to figure out what topic a specific document is mainly talking about. In other words, we should find

out the topic node from the RDF graph. The RDF document created based on *linked data* [11] (e.g., the document from DBpedia [12]) usually contains property *p:primaryTopic* and under this circumstance, the value of this property will be recognized as the topic node. However, when the RDF document does not contain this kind of properties, we should find a generic method to make up. Usually, the centrality of a node in an undirected graph is calculated by counting the number of connections it has. Since RDF graph is a directed graph, a simple and intuitive method for finding out the central node is counting the sum of the in-degree and the out-degree for each node, which is actually the occurrence of a node. In [13], Xiang et al. compared five measurements used for automatically summarizing ontologies and the experiments show that the weighted in-degree centrality measures have the best performance. However, for the case that the target RDF documents not only contain ontologies but also contain the large amount of RDF individuals, this measurement usually does not work effectively. Moreover, according to the definition of the inverse property, each property can have its own inverse property. Therefore, for each RDF node, its in-degree and our-degree should be equivalently important in theory. We can learn this well through the following instance.

Suppose there is an RDF document which is mainly talking about person P_A and its corresponding graph is described in Figure 2. In this figure, we can see that person P_A knows person P_B, person P_C and person P_D. Moreover, P_A, P_B, P_C, P_D and P_E are all working at the same firm. Apparently, the occurrence of node $Firm$ is larger than that of node P_A. However, we can not draw the conclusion that node $Firm$ is the topic node in this graph. Actually, this document is mainly talking about person P_A. Therefore, just based on the in-degrees and the out-degrees of nodes, we can not determine the topic node. Here, we present our heuristic method that makes use of the original URLs of RDF documents to find out the topic nodes more effectively. Based on our large amount of observations, more than 90% of RDF documents have the topic-related information hidden in their URLs. For instance, URL *http://dblp.l3s.de/d2r/resource/authors/Tim_Berners-Lee* contains the name of the topic-related person *Tim Berners Lee*. Therefore, instead of selecting one node with the max occurrence, we choose several topic-node candidates and compare their URIs to the original URLs of the RDF documents. Then we regard the candidate with the max similarity as the topic node. Since URIs and URLs are all strings and each string can be recognized as a set of characters or digits, we calculate the similarity of string α and string β by the following formula:

$$similarity(\alpha, \beta) = \frac{\alpha \cap \beta}{min(|\alpha|, |\beta|)}$$

Here, $|\alpha|$ and $|\beta|$ denote the length of string α and string β respectively. In other words, the similarity of α and β is the percentage the length of their longest common substring is of the length of the shorter string. In order to alleviate the influence of the nodes with the high similarity but the low occurrence on the performance of our topic-node selection, we give a method to exclude them. Firstly,

we find out the node with the max occurrence. Secondly, we calculate the percentage the occurrence of each node is of this max occurrence. Then we compare the percentages with a predefined threshold and take off the nodes whose percentages are lower than this threshold. Finally, we get a set of candidate nodes and then calculate the similarities between each of their URIs and the original URL of the RDF document. The candidate node with the largest similarity will be regarded as the topic node. Indeed, it is not uncommon that an RDF document contains more than one topic node, especially when this document has a large amount of triples. Under this circumstance, takeing the efficiency into account, our method will still return one topic node as the clue.

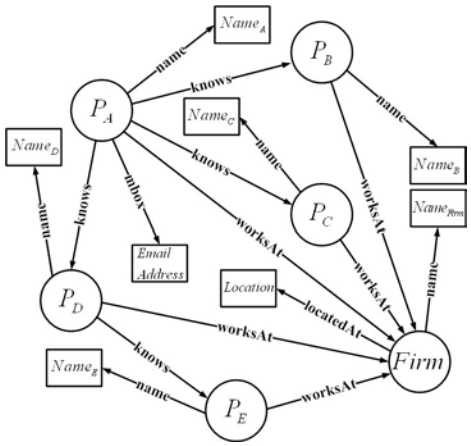


Fig. 2. Graph of an RDF document

We further generate the snippet that contain the topic node. We first calculate the occurrence of each property connected with this node. There are two types of properties: object property and datatype property [15]. Suppose the topic node is the subject of the statement, our algorithm returns the types of the corresponding objects using diverse ways for different cases based on the reasoning technique described in Section 3. For the case that the property is an object property, we first try to find out the type of the object defined in this document. If we find it, it is then regarded as the type of the object; if we can not find it, we further check if the indexed content, composed of the triples in the original RDF document, the inferred triples and the triples from the ontologies, has already contained this type definition or retrieve and index other new RDF documents if necessary. Likewise, for the case that the topic node is the object of the statement, we can also use the above method to find the type of the corresponding subject.

5 RDF Statements Ranking Algorithms

In this section, we propose two algorithms for ranking the topic-related statements and the query-related statements respectively.

5.1 Ranking of Topic-Related Statements

Usually, the topic-related statements occupy too many lines in the final snippet. We should find out an automatic method for selecting the most important statements and display them at the top of the snippet. In this subsection, we propose an algorithm for ranking the topic-related statements based on the priorities of the properties, as shown in Algorithm 1.

Algorithm 1: Topic-related statements ranking algorithm

```
Input: Unranked snippet S and property table T
Output: Snippet S' after ranking

begin
    Set all the values in Display column to "SHOW";
    for each row r ∈ T do
        add the corresponding URI of r to pro_uri;
        if S does not contain pro_uri then
            Set the Display value of the current property to "EXCLUDED";
            continue;
        end
        add the values of the Exclusive column to an array ex_pro_uris and
        add the values of the Relative column to an array re_pro_uris;
        for each string epu ∈ ex_pro_uris do
            Set the Display value of epu to "EXCLUDED";
        end
        for each string rpu ∈ re_pro_uris do
            if the Display value of rpu ≠ "EXCLUDED" then
                set the Display value of rpu to "SHOW";
                revise the priority of rpu based on the current property;
            end
        end
    end
    add the URIs whose corresponding Dispaly value are "SHOW" in T to an
    array pro_uris;
    sort the values in pro_uris ascendingly;
    for each string spu ∈ pro_uris do
        get the statements containing spu and add them to S';
    end
end
```

We collect a variety of RDF schemas and finally find that different properties have different importance degrees in each RDF schema. For instance, for the FOAF [16] file case, property *foaf:name* is more important than other properties. Therefore, we set different priorities to different properties. The more important the property is, the higher its corresponding priority is. Here, we also consider the relationships between properties. Generally speaking, there are two types of relationships: *correlative* and *exclusive*. If a specific property appears in the snippet and some other ones should also appears, these two properties are *correlative*. On the other hand, if a specific property appears and some other ones should be hidden in the snippet, these two properties are *exclusive*. For instance, in the

FOAF file, *foaf:surname* and *foaf:family_name* are correlative properties. However, *foaf:name* is exclusive to *foaf:surname* and *foaf:family_name* since users do not expect to see duplicated information in the final length-limited snippet. For each property, we define its priorities, correlative properties and exclusive properties, and finally save them into a table.

5.2 Ranking of Query-Related Statements

As a matter of fact, users usually care about the RDF documents mainly talking about the individuals which are associated with the users' inputs. Therefore, if the topic of a specific RDF document apparently has nothing to do with the user's input, we should tell him or her the relations between them. Here, we propose a method for ranking the statements in the query-related section. The user's input is not a node but a string with various formations. For instance, if a user expects to look for the information about *Tim Berners Lee*, he or she may input "TIM BERNERS LEE" or "Tim Berners-Lee". Here, we use Lucene[1] to split the query phrase into separate words and uniform the user's input and then select the nodes associated with these inputted words.

Table 1. Statement sequence after ranking

Rank	Statements
1	Query-Related Node + Predicate + Topic Node
2	Topic Node + Predicate + Query-Related Node
3	Query-Related Node + Predicate + Un-Topic Node
4	Un-Topic Node + Predicate + Query-Related Node
5	Topic Node + Predicate + Un-Query-Related Node
6	Un-Query-Related Node + Predicate + Topic Node

We also give the criteria for deciding if a node is a query-related one or not. If a node denotes an individual and its URI contains the words the user has inputted, the node is a query-related node. If the node denotes a literal (e.g., string) and it contains the words the user inputted, the node is a query-related node. Otherwise, it is not a query-related node at all. After the selection, we list all the statements containing the topic node or the query-related nodes according to the sequence described in Table 1. It is also notable that the statements which have been displayed in the topic-related section will not be displayed in the query-related section any more.

6 Descriptions Generation

An RDF statement contains the subject, the predicate and the object and sometimes it is not readable since the predicate is usually personally defined for short. In this section, we give our method for generating more understandable descriptions from RDF statements by splitting the predicate in a reasonable way.

[1] `http://jakarta.apache.org/lucene`

Algorithm 2: Description generation algorithm

```
Input: RDF statements Stats and pre-indexed content Con_indexed
Output: discription Sents in the snippet
begin
    for each statement s ∈ Stats do
        create strings sub_part, pred_part and obj_part;
        add the subject of s to Resource subject;
        add the predicate of s to Property predicate;
        add the object of s to RDFNode object;
        if subject's type ≠ null then
            sub_part = subject's type name + subject's identifier;
        end
        else
            if subject has not been indexed in Con_indexed then
                retrieve and index the RDF document with the URI of subject;
            end
            else
                find the type of the subject in the index;
                sub_part = subject's type name + subject's identifier;
            end
        end
        split the name of predicate and add them to pred_part;
        if object is an instance of Resource then
            generate Resource obj_resource as the copy of object;
            if obj_resource is an individual && obj_resource's type ≠ null then
                obj_part = object's type name + object's identifier;
            end
            else
                if object has not been indexed in Con_indexed then
                    retrieve and index the RDF document with the URI of
                    object;
                end
                else
                    find the type of the object in the index;
                    obj_part = object's type name + object's identifier;
                end
            end
        end
        else
            generate Literal obj_literal as the copy of object;
            obj_part = "Literal␣" + "\"" + obj_literal's lexical content + "\"";
        end
        add sub_part+"␣"+pred_part+"␣"+obj_part to Sents;
    end
end
```

Based on the analysis by Xiantang and Chris [17], we classify the labels of the predicated into two sorts: single-word predicate and multi-word predicate. For the former case, there is no need to split it further; for the latter case, tradition-

ally, each word will be separated by a specific symbol such as the underline, the horizontal line and so on. Here, we use regular expressions to generate a predicate pattern for splitting the predicates. It is also possible that a predicate dose not contain any separators but its first letter is in upper case. Then we split the predicate using the uppercase letters as the separators. We also return the types of the subject and the object to make the final description more understandable. However, maybe the processed RDF document does not contain the type declaration of each individual or literal. Based on the method described in Section 4, we use current indexed content or retrieve another new RDF document to find the type if necessary. The algorithm for generating description is described in Algorithm 2. Usually, the URIs of the resources are long strings and we should not return them directly to users since some URIs do not end with their names (e.g., URIs of anonymous resources) since this kind of URIs will make users puzzled if they appear in the final snippets. Here, we use the reasoning techniques described in Section 3 again to find the missed types resources.

7 Personalized Snippet Generation

Considering that users may come from different language regions, in this section, we introduce our method for internationalizing the generated snippets. Based on our snippet generation framework, each snippet will finally fall into 2 parts: the topic-related section and the query-related section. The topic-related section is composed of the topic node and the statements that take the topic node as the subject or the object. The query-related section is composed of all the statements which are displayed according to the sequence described in Table 1. For each of the most popular languages around the world, we create the corresponding template for the unchangeable content in the snippets and use variables to represent the changeable content. For instance, in the topic-related section of the English template, the unchangeable content is "This RDF Document mostly talks about the". Finally, we replace the variables with the content generated by our predefined multi-language template. It is also notable that Sindice will display the snippets in a specific language the user chooses at the top of the RDF document lists.

Moreover, since the users' requirements for finally displayed snippets are usually different, we generate the personalized snippets according to their predefined preferences. For instance, users can set the configuration about how many lines should be finally displayed. In our approach, we manage the data and the formation of the snippets separately, which will be stored in the XML file and the XSLT [18] file respectively.

8 Experiments and Performance Evaluation

Our snippet generation approach has been implemented and integrated using Java and Ruby on Rails (ROR). The data set is provided by Sindice which currently indexes over 26.69 millions RDF documents. Suppose the inputted

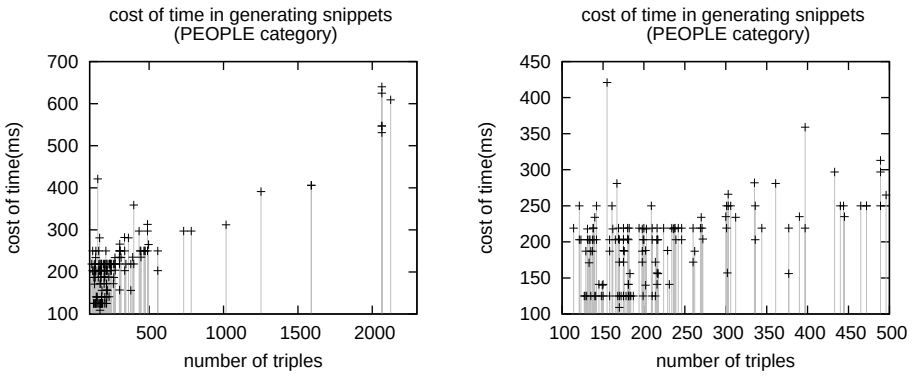
querying phrase is "Tim Berners Lee", Figure 3 shows the snapshot of the generated snippets. From the snippet belonging to RDF document with the URL *http://dblp.l3s.de/d2r/resource/authors/Tim_Ber-ners-Lee*, we can see that this document is mainly talking about *Tim Berners-Lee*, which is consistent with the querying phrase. Below the topic description, more information about this topic are further displayed based on the predefined priorities of properties. We also use "see all" button to shorten the length of each line. By pushing this button, users can get the whole version of the snippet. Since the querying phrase exactly matches the topic node, the query-related section dose not exist in the final snippet of this document.

Fig. 3. Snapshot of generated snippets

From the snippet of another RDF document with the URL *http://www.mindswap.org/2003/submit-rdf/936.rdf*, we can see that the extracted topic node with the type *Unpublished* actually denotes an article, which is apparently not associated with the inputted querying phrase. In this case, the query-related section

will be generated according to the sequence described in Subsection 5.2. From this section, we can see that *Tim Berners-Lee* is one of the authors of this article. So users do get extra and helpful information from this section. Moreover, users can recursively get further snippets of the resources which exist in the current snippet by pushing the buttons "[+]" behind the resources.

In order to experiment with our topic node generation algorithm, we input 54 different querying phrases into Sindice using *keyword search*, covering people, locations, Web techniques, marvelous spectacles, organizations and sports, and get totally 1152 indexed RDF documents. These documents contain a variety of RDF formations (e.g., FOAF and SIOC [19]) coming from diverse sources (e.g., Wikipedia [20] and DBpedia). Firstly, we use the max in-degree method, the max out-degree method and the simplex occurrence method to look for the topic node, respectively. Secondly, we use our max occurrence method associated with the original URL to generate the topic node again. Then we ask six domain experts to manually select the topic nodes from the indexed RDF documents with the help of the Tabulator[2]. Assuming the manually-found topic nodes are all correct and reasonable, we compare our method with the aforementioned methods. Finally, we find that the max in-degree method has the lowest accuracy and the two occurrence-based methods both work better than the max out-degree method. Moreover, our original-URL-based method works better when the RDF documents contain a the large amount of RDF triples. Out of the above 1152 RDF documents, 1072 correct topic nodes are finally generated using our heuristic topic discovery method and the accuracy rate is 93.1%.



(a) 0∼2300 triples (b) 0∼500 triples

Fig. 4. Cost of time in generating snippets from *People* category

We also evaluate the performance of our whole snippet generation approach. Within the generation process, for each RDF document, we record the cost of time and the number of triples contained by this document. Figure 4 describes

[2] `http://www.w3.org/2005/ajar/tab`

the cost of time in generating snippets from the *People* category. According to subfigure (a), the cost of time increases with the increasing of the number of triples apparently. Actually, the number of the RDF documents containing less than 500 triples accounts for 89.7% of the total documents. From subfigure (b), just taking the RDF documents contain less than 500 triples into the consideration, we can see that the time of cost is not associated with the number of triples any more, especially for the documents containing less than 250 triples. Likewise, for other categories, we calculate the percentage the number of less-than-500-triples documents is of the total number of documents and the average costs of time in generating snippets from less-than-500-triples document and all the documents, respectively. The results are shown in Table 2. From this table, we can see that the average cost of time for less-than-500-triples documents is less than $0.2s$. So for most of the searched RDF documents, our approach can return their snippets quickly.

Table 2. Experiment results

category	percentage$_{<500}$	average $COT_{<500}$	average COT_{total}
People	89.7%	198.2 ms	616.1 ms
WebTech	100.0%	186.9 ms	186.9 ms
Organization	99.3%	183.7 ms	201.1 ms
Spectacle	86.3%	198.4 ms	466.3 ms
Location	98.8%	187.9 ms	188.9 ms
Sport	100.0%	193.2 ms	193.2 ms
Average	94.8%	191.0 ms	353.3 ms

9 Related Works

The motivation for summarizing the ontologies is that the current formations for storing ontologies are difficult for non-technician users to understand. Some work for verbalizing ontologies have been done recently.

Wilcock described the on-going work on an ontology verbalizer which combines the Semantic Web techniques with natural language generation and text-to-speech in [1]. Since this verbalization is mainly based on XSLT, it can not deal with all the kinds of RDF documents but focuses on some specific kinds. Kalina et al. proposed a method of automatically generating reports from domain ontologies encoded in OWL using MIAKT generator [2], which takes the medical ontology, RDF description of the case and the MIAKT natural language generation lexicon as the input. Daniel et al. gave an algorithm for using a part-of-speech (POS) tagger, a fast and simple application, to produce concise, accurate natural language paraphrases for OWL concepts [3]. Shumao et al. proposed a Model Driven Integration Architecture (MDIA) to integrate rigorous model specifications and generate context-aware application either semi-automatically or automatically [4]. Chris et al. resented the evidence that natural language

words are used in complex ways in current ontologies and gave their own work using natural language generation to present parts of ontolgoies [5]. Based on their analysis, Gunther et al. generated a proposal for linguistically determined label generation which benefits the process of mapping OWL concepts to natural language patterns [6].

Fresnel [14] is a display vocabulary for specifying how RDF graphs are presented. However, it is tedious and time consuming that users are required to know how to create the lenses and the formats, which are both the important components for generating the descriptions. In [13], the experiments show that the weighted in-degree centrality measures have the best performance in finding out the most content. However, this measurement usually does not work effectively when the target RDF documents not only contain ontologies but also contain the large amount of RDF individuals. To the best of our knowledge, topic-node discovery and RDF statements ranking have not been further researched and still faces difficulties nowadays. Based on our work, users can conveniently get the snippets without any necessary domain knowledge.

10 Conclusions and Future Work

This paper proposes an automatic approach for generating snippets from RDF documents. This approach has been already implemented and integrated into the Sindice demo website[3] for users to test our algorithms. Our heuristic topic-discovery method can find out the topic node efficiently and effectively, according to the occurrences of RDF nodes and the original URLs of the RDF documents. Moreover, two ranking algorithms, topic-related-statements ranking and query-related-statements ranking, are presented in order to make the generated snippets more understandable. The use case of our approach is also given in the end and the experimental results indicate the superiority and high efficiency of our approach.

Our long-term goal is to bring certain simple but effective natural language processing techniques into the snippet generation process to improve the readability of the snippets. Besides, some RDF documents probably have multiple topics and it seems more reasonable to list the most promising topic nodes. So we need a way to rank all the possible topic nodes using a more effective and efficient method that occupies relatively less computing resources.

Acknowledgement

This work is supported by OKKAM under the Grant No. 215032. We would like to thank Gabriele Renzi for his work on the maintenance of the snippet generation demo and Michele Catasta, Richard Cyganiak, Holger Stenzhorn and Adam Westerski for their valuable suggestions and efforts on experiments.

[3] `http://beta0.sindice.com`

References

1. Wilcock, G.: Talking OWLs: towards an ontology verbalizer. In: Proceedings of Human Language Technology for the Semantic Web and Web Service Workshop at the International Semantic Web Conference, pp. 109-112 (2003)
2. Bontcheva, K., Wilks, Y.: Automatic report generation from ontologies: the MIAKT approach. In: Proceedings of the International Conference on Applications of Natural Language to Information System, pp. 324-335 (2004)
3. Hewlett, D., Kalyanpur, A., Kolovski, V., Wiener, C.H.: Effective natural language paraphrasing of ontologies on the Semantic Web. In: Proceedings of the End User Semantic Web Interaction Workshop at the International Semantic Web Conference (2005)
4. Ou, S., Georgalas, N., Azmoodeh, M., Yang, K., Sun, X.: A model driven integration architecture for ontology-based context modeling and context-aware application development. In: Proceedings of the European Conference on Model Driven Architecture Foundations and Applications. LNCS, vol. 4066, pp. 188-197, Springer, Heidelberg (2006)
5. Mellish, C., Sun, X.: The Semantic Web as a linguistic resource: opportunities for natural language generation. Knowledge Based Systems 19(5), 298-303 (2006)
6. Fliedl, G., Kop, C., Vöhringer, J.: From OWL class and property labels to human understandable natural language. In: Proceedings of the Interational Conference on Applications of Natural Language to Information Systems. LNCS, vol. 4592, pp. 156-167, Springer, Heidelberg (2007)
7. Harth, A., Hogan, A., Delbru, R., Umbrich, J., O'Riain, S., Decker, S.: SWSE: answers before links! In: Proceedings of the 6th Semantic Web Challenge (2007)
8. Tummarello, G., Delbru, R., Oren, E.: Sindice.com: weaving the open linked data. In: Proceedings of the International Semantic Web Conference. LNCS, vol. 4825, pp. 552-565, Springer, Heidelberg (2007)
9. Oren, E., Delbru, R., Catasta, M., Cyganiak, R., Stenzhorn, H., Tummarello, G.: Sindice.com: a document-oriented lookup index for open linked data. International Journal of Metadata, Semantics and Ontologies 3(1) (2008)
10. ter Horst, H.J.: Combining RDF and part of OWL with rules: semantics, decidability, complexity. In: Proceedings of the International Semantic Web Conference. LNCS, vol. 3729, pp. 668-684, Springer, Heidelberg (2005)
11. Berners-Lee, T.: Linked data (2006), available at `http://www.w3.org/DesignIssues/LinkedData.html`
12. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: DBpedia: a nucleus for a Web of open data. In: Proceedings of the International Semantic Web Conference. LNCS, vol. 4825, pp. 722-735, Springer, Heidelberg (2007)
13. Zhang, X., Cheng, G., Qu, Y.: Ontology summarization based on RDF sentence graph. In: Proceedings of the 16th International Conference on World Wide Web. pp. 707-716, ACM Press (2007)
14. Lee, R.: Introduction to Fresnel: an RDF display vocabulary (2006), available at `http://dig.csail.mit.edu/2006/Talks/0724-fresnel/`
15. Manola, F., Miller, E.: RDF primer (2004), available at `http://www.w3.org/TR/REC-rdf-syntax/`
16. Brickley, D., Miller, L.: FOAF Specification (2007), available at `http://xmlns.com/foaf/0.1`

17. Sun, X., Mellish, C.: An experiment on "free generation" from single RDF triples. In: Proceedings of the 11th European Workshop on Natural Language Generation. pp. 105-108 (2007)
18. Clark, J.: XSL Transformations (XSLT) (1999), available at `http://www.w3.org/TR/xslt`
19. Berrueta, D., Brickley, D., Decker, S., Fernández, S., Görn, C., Harth, A., Heath, T., Idehen, K., Kjernsmo, K., Miles, A., Passant, A., Polleres, A., Polo, L.: SIOC core ontology specification (2007), available at `http://rdfs.org/sioc/spec/`
20. Völkel, M., Krötzsch, M., Vrandecic, D., Haller, H., Studer, R.: Semantic Wikipedia. In: Proceedings of the 15th International Conference on World Wide Web. pp. 585-594, ACM Press (2006)