WEB-BASED ONTOLOGY EDITOR ENHANCED BY PROPERTY VALUE EXTRACTION

Takahiro Kawamura[1], I Shin[1] and Akihiko Ohsuga[1]

[1]Graduate School of Information Systems, University of Electro-Communications
1-5-1 Chofugaoka, Chofu-shi, Tokyo 182-8585, Japan

ABSTRACT

Linked Open Data and its consuming services are increasing in the areas of electric government, bio-science researches and smart community projects. Therefore lightweight ontologies used for those areas are also becoming important. This paper proposes a web-based ontology editor that is an ontology building service especially for the lightweight ontologies. It is designed not only for ontology experts, but also for users and/or developers of consumer services, that is, non-experts of the ontology. Thus we focused on ease of use, no installation, cooperative work environment, and also provided sample applications to keep the users' motivation. Furthermore, it offers a function that extract pairs of <property, value> belonging to a certain instance of a class. First, we introduce the design and the implementation of our ontology editor, and then present the extraction mechanism of <property, value> pairs and confirms its accuracy in experimental evaluations.

KEYWORDS

Ontology Editor, Property Value, Bootstrapping, Dependency Parsing

1. INTRODUCTION

Recently, Linked Open Data (LOD) and services that are using the LOD have increased in the areas of electric government, bio-science researches and smart community projects. In those areas, most of ontologies used in the LOD have simpler structures in comparison with traditional ontologies for manufacturing design and medical diagnosis, and so forth. Faultings [1] said that "is-a" relation makes up 80-90% of all the relations in such lightweight ontologies. However, ontology building tools such as protégé [2] are mainly for the ontology experts, and few of them are focusing on developers and users of consumer services. Therefore, we have developed a web-based ontology editor that is an ontology building service aiming at offering an environment by which non-experts are able to make necessary ontologies for their purposes. The target users of this editor are people who have little expertise about what are ontology, its schema, organization, and technical terms. Or, people who have some knowledge about the ontology, but no experiment of building ontology are also the targets. Our editor tried to solve the following three problems that the target users may encounter when using the ontology tools. We had several interviews with non-ontology developers and programmers and summarized as follows:

1. Necessary tool preparation and its usage are unclear because of many technical terms and functions.
2. It is hard to know what term should be input as classes, instances, and properties, since they have never thought about things according to the ontological method.
3. It is problematic to register a large amount of the instances and property values (this problem is shared with the expert and the non-experts).

DOI : 10.5121/ijwest.2013.4301

To solve these problems, we took the following approaches:

A. Easy preparation for introduction.
Web-based editors do not need the installation of any tools, and can be used by web browsers.

B. Improvement of usability.
We limited use of the technical terms, and narrowed the functions only for browsing and editing of the lightweight ontology. Also, we tried to keep usability even in the browser application.

C. Support of term registration.
It would be almost impossible that a single person can register every instance and property in a large domain. Therefore, our editor extracts candidates of the property values from the Web, and recommends them to the user.

D. Keeping of motivation.
The editor also offers sample applications to show what services can be made by using ontologies. Moreover, it opens web service APIs to access the stored ontologies to be used by external services.

E. Support of collaborative work.
Multiple access to a single ontology is possible, thus the ontology can be built by a team.

The above three problems would not be solved by any single approach, and have 1-to-n relations to the five approaches. Fig.1 shows the relations between the problems and approaches we have assumed. Furthermore, their effect measurement for each problem would not be necessarily verified quantitatively, because of the large dependency of user's feeling and sensitivity. Therefore, this paper firstly describes overview of our service, and then we focus on the approach "C. Support of term registration" with details of the implementation and the evaluation. Although the approach C is corresponding to the problem 3 and 2, we believe that an accuracy of recommended terms can be regarded as a metric to measure the degree of the user support for these problems.

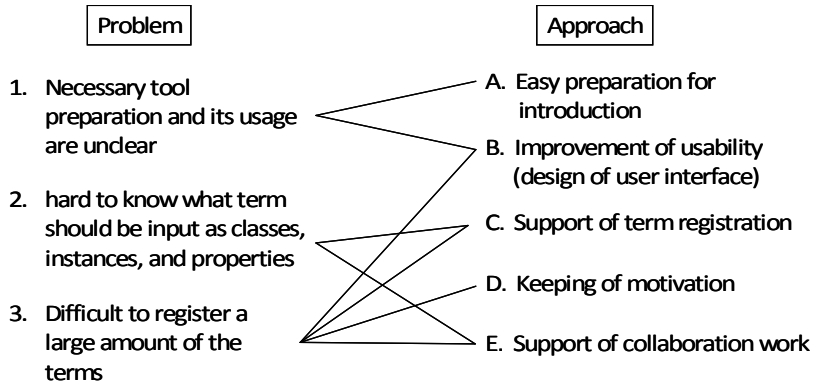


Figure 1. Problems and Approaches

The outline of this paper is as follows. We first describe the service overview with its interface and functionality in section 2. Then section 3 shows the <*property*, *value*> extraction mechanism and its evaluation. In section 4, we discuss the limitations and the improvements of our proposed approaches. Finally, we show the related works in section 5, and conclude this paper in section 6.

2. DESIGN AND IMPLEMENTATION OF ONTOLOGY EDITOR

The service logically consists of a front-end (Flash website) and a back-end (web services) (Fig.2). The front-end has an ontology editing function, the recommendation function, and two sample applications. Then, the back-end provides the APIs to access the ontologies built by the service so that other systems can use the ontologies. We connected the front- and back-end by Asynchronous Flash, and realized high responsibility and operability as a browser application. This section describes the editing function, the sample applications and implementation issues.

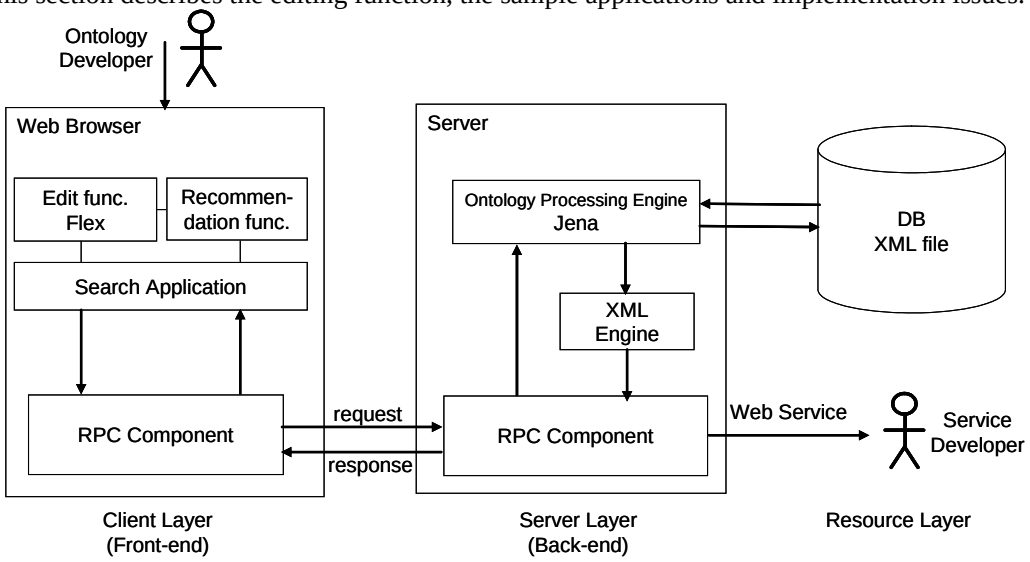
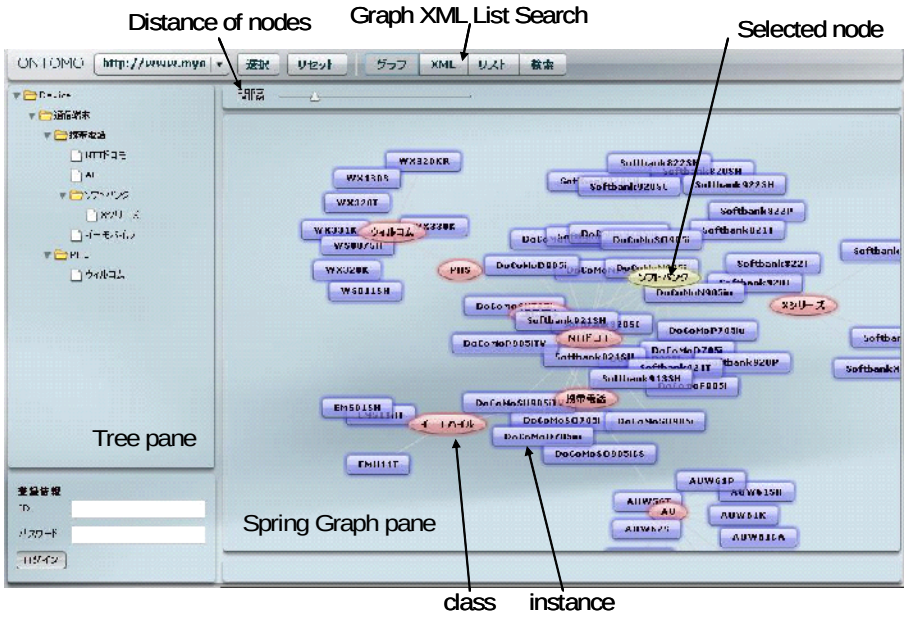


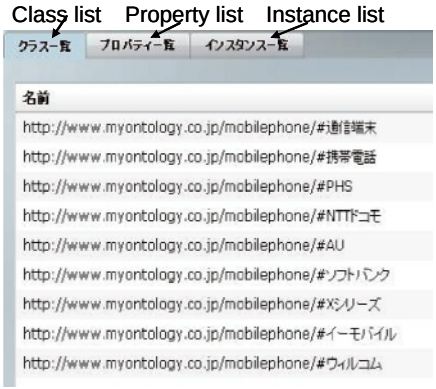
Figure 2. Service Architecture

2.1. Editing Function

The editing function is for the lightweight ontology, and has two roles: browsing major components of the ontology such as the classes, instances and properties, and basic functions like new addition, edit, and delete from them.

We provide three ways of the ontology browse. The first one is a Graph view to intuitively figure out the ontology structure. Fig.3 top indicates that Flash and Spring Graph that is an automatic graph allocation library based on a spring model visualize a hierarchical structure composed of the classes, subclasses and instances. It can adjust, for example, a distance of an instance and a class. In addition, if the user moves a class, then instances of the class automatically follows and are re-arranged with the pre-defined distance. Also, if the user double-clicks a class, the instances will hide, and vice versa.





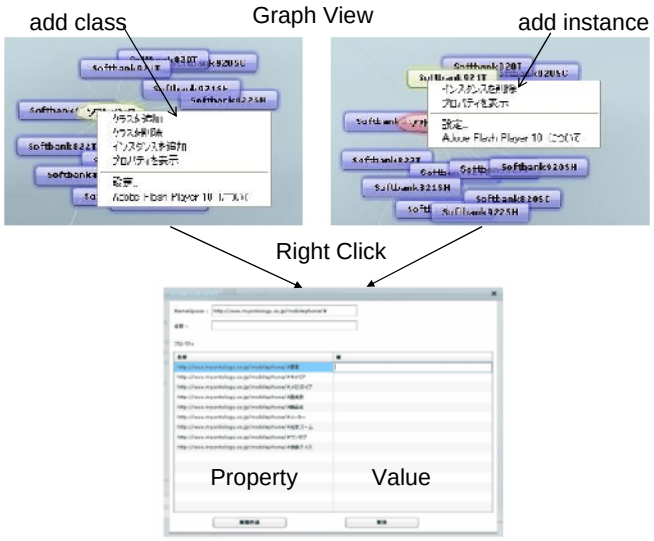


Figure 3. User Interface

On the other hand, the Graph view is unsuited to read through a list of the instances, and check the details of their definitions. Therefore, we added a List view and an XML view. The List view (Fig.3 middle) has three lists for the classes, properties, and instances. The XML view shows OWL [3] data of the ontology as follows:

```
<rdf:RDF xmlns:mobilephone=http://www.myontology.co.jp/mobilephone/#
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd=http://www.w3.org/2001/XMLSchema#
    xmlns:rdfs=http://www.w3.org/2000/01/rdf-schema#
    xmlns:owl=http://www.w3.org/2002/07/owl#
    xmlns:daml="http://www.daml.org/2001/03/daml+oil#">
    <owl:Ontology>
      <rdfs:comment>MobilePhone OWL Ontology</rdfs:comment>
    </owl:Ontology>
    <owl:Class rdf:about="http://www.myontology.co.jp/mobilephone/#softbank">
      <rdfs:subClassOf>
        <owl:Class rdf:about="http://www.myontology.co.jp/mobilephone/#mobile phone"/>
      </rdfs:subClassOf>
    </owl:Class>
</rdf:RDF>
```

Ontology editing starts with a popup menu that appears with a right click on Graph view (Fig.3 bottom). If the user does the right click on a class or instance, the popup menu with “addition”, “edit”, and “delete” will appear. Furthermore, the ontology is stored in the DB in the back-end, and can be exported in OWL files, or accessed by other services.

2.2. Sample Applications

The service also provides sample applications, one of which is a product search application. In this application, the user can find products that have specified properties. We prepared the product ontologies of three domains in advance: mobile phones, digital cameras, and media players like iPod. If the user searches any products with a common property like "SD card", the products with the property will be found across the three domains. It is also possible to search with multiple properties like hours of continuous operation and TV function (1 seg), and then the ontologies of the mobile phones and the media players will be searched.

Moreover, a blog search application shows the related blogs on the product found by the above application by using Google blog search, since the latest word-of-mouth information about the product would be useful for the user.

2.3. Implementation

Our editor is a three-tier system (Fig.2), where a client layer provides a flexible user interface using Adobe Flex [4]. Then, a server layer has an ontology processing engine with Jena [5], and a resource layer stores the ontology data in an XML format in MySQL. Asynchronous call between the client layer and the server layer is realized by Flex components like Flex Remote Object, Flex Data Service. When the user operates the views with Flash, the client layer sends a request to the server layer, and the Flex component calls corresponding APIs. The result is returned in an XML format for the Graph or the List to the client layer, and then the XML engine transforms it to show in the Flash view.

3. PROPERTY VALUE EXTRACTION

3.1. Extraction Mechanism from the Web

In the practical work of the instance addition, it would be difficult to register every detail of each instance without any help. Especially, although the instance names can be collected from a list on

any summarized site and the necessary property names would be defined by the users based on their service requirements, values of all the properties for each instance would need the help of the service. Therefore, we developed the function to extract values of each property of the instances from the Web, whereas the user finally selects and adds some of them as new correct <*property, value*> pairs. This function involves a bootstrapping method [6] and a dependency parsing based on [7].

The Process of the extraction is as follows (Fig. 4). First of all, we make a keyword list that includes an instance name and a logical disjunction of property names, and then search on Google, and receive the result list, which includes more than 100 web pages. Next, we collect the pages except for PDF files and also check Google PageRank for each page.

As the bootstrapping method, we first extract specific patterns of the DOM tree from a web page based on the keys that are the property names (and their synonyms), and then we apply that patterns to other web pages to extract the values of the other properties. This method is mainly used for the extraction of <*property, value*> pairs from structured parts of a page such as tables and lists (Fig. 4 left).

However, we found there are many unstructured sites that are explaining the contents only in plain text. Therefore, we developed an extraction method using the dependency parsing, since a triple <*instance, property, value*> corresponds to <*subject, verb, object*> in some cases. It first follows modification relations in a sentence from a seed term that is an instance name or a property name (and their synonyms), and then extract the triple, or a triple like <−, *property, value*> in the case of no subject in the sentence. See Fig. 4 right.

We then combine all the property values obtained by the above methods, and select the values that match to co-occurrence strings with the corresponding property names. A set of the co-occurrence strings is prepared in advance, for example, the property “price” must obviously co-occur with a string of JPY, USD or others. Then, we form some clusters of the same property values for each property name based on LCS (Longest Common Substring). Furthermore, for correction of errors that include not only errors of the extraction, but also the information in the source pages, we sum up the Google PageRank of the source pages for each cluster to determine the best possible property value and the second-best. Finally, the user determines a correct value from the proposed two candidates.

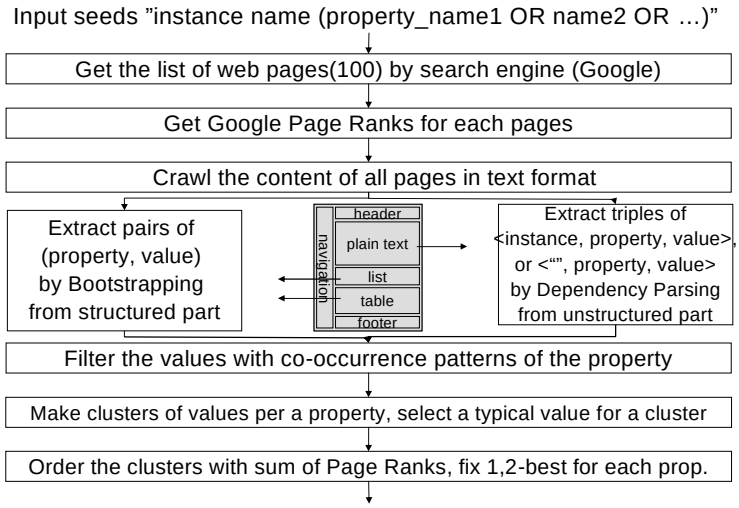
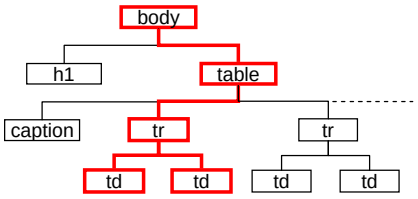


Figure 4. Process of LOD content generation

1. Key match to original structure

```
<body>
<h1>begonia</h1>
<table>
<caption>Characteristics</caption>
<tr><td>Color</td><td>red</td></tr>
<tr><td>Light</td><td>Part Shade</td></tr>
<tr><td>Water</td><td>Normal</td></tr>
</table>
```

2. DOM pattern extraction



3. Other (property, value) pairs are extracted

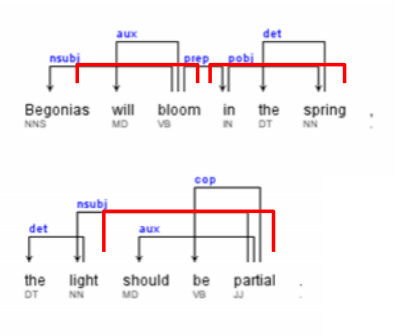
(Light, Part Shade)
(Water, Normal)

1. Seed match to original sentence

"***Begonias*** will bloom in the spring,

the ***light*** should be partial."

2. Dependency parse



3. Triples <plant name, property, value> are extracted

<Begonia, bloom, spring>
< - , Light, partial>

Figure 5. Examples of bootstrapping and dependency parsing

3.2. Evaluation of Property Value Extraction

Table 2. Extraction accuracy

Accuracy(%)	1-best	2-best	1-best (bootstrap only)	1-best (dependency only)
Precision	85.2	97.4	88.6	85.2
Recall	76.9	87.2	46.2	76.9
Amount Ratio (%)	–	–	10.8	89.2

We applied this mechanism of <*property*, *value*> extraction to collect the values of 13 properties for 90 products (garden plants). The result shown in Table 2 includes an average precision and recall of the best possible value (1-best) obtained through the whole process, the bootstrapping method only, and the dependency parsing only, and then those of the second possible value (2-best). It should be noted that we collected 100 web pages for each product, but some reasons such as DOM parse errors and difference of file types reduced the amount to about 60%. Properties like product description, which are not clear whether it is true or not, were out of scope of this evaluation. If there are more than two clusters whose sums of the PageRank are the same, we regarded them all as the first position. The accuracy is calculated in units of the cluster instead of each extracted value. That is, in the case of 1-best, a cluster which has the biggest PageRank corresponds to an answer for the property. In the case of 2-best, two clusters are compared with the correct value, and if either one of the two answers is correct, then it is regarded as correct (thus, it is slightly different than average precision).

$$N - best\ precision = \frac{1}{|D_q|} \sum_{1 \le k \le N} r_k$$

, where $|D_q|$ is the number of correct answers for question q, and *rock* is an indicator function equaling 1 if the item at rank k is correct, zero otherwise. The bootstrapping method only and the dependency parsing only mean to form the clusters out of the values extracted only by the bootstrapping and the dependency parsing, respectively. A cluster consists of the extracted values

for a property, which seem identical according to LCS, but the number of the values in a cluster may vary from more than 10 to 1. Finally, if there are multiple correct values for a property, we selected the most dominant one.

The best possible values (1-best) achieved an average precision of 85% and an average recall of 77%. However, the 2-best achieved an average precision of 97% and an average recall of 87%. Therefore if we are permitted to show a binary choice to the user, it would be possible to present a correct answer in them in many cases. The accuracy of the automatic generation would not be 100% over all, and then a human checking is necessary at the final step. The binary choice would be a realistic option.

In detail, the bootstrapping collects smaller amounts of values (11%), so that the recall is substantially lower (46%) than the dependency parsing, although the precision is higher (89%). It is because data written in the tables can be correctly extracted, but lacks diversity of properties. Semantic drift of the values by generic patterns that is a well-known problem in the bootstrapping method did not happen in this case, because target sources are at most top 100 pages of the Google result, and the values are sorted by the PageRank at the end.

On the other hand, the dependency parsing collects a large amount of values (89%), although it was a mixture of correct data and noises. However, the total accuracy is affected by the dependency parsing, because the biggest cluster of the PageRank is composed mainly of the values extracted by the dependency parsing. Therefore we need to consider weight on the values extracted by the bootstrapping.

4. DISCUSSION

In terms of “C. Support of term registration” in section 1, we presented the *<property, value>* recommendation mechanism to reduce the user's strain. As the result, it achieved the high precision although the recall is relatively low. However, it would be a difficult task for the target users of this editor who are non-experts of the ontology, to extract only the correct terms from a large set of the terms (Of course, it depends on their domain knowledge). Meanwhile, in case of the low recall the only thing that the user needs is to repeat the process of the extraction with the different seeds. That is obviously a simpler task.

In terms of other approaches: “A. Easy preparation for the introduction” and “B. Improvement of usability” as mentioned in section 1 we eliminated the tool installation by making it the browser application, and realized operability like a desktop application by using Adobe Flex. Also, the Spring Graph realized visibility of the whole ontology. In terms of “D. Keeping of motivation”, we tried to motivate the user by presenting the sample applications and open web APIs, and for “E. Support of collaborative work” an exclusive access control of ontology DB enabled multiple developers to build an ontology at the same time. For those approaches, however, there are some other methods like the improvement of documents, more innovative user interfaces, and a point system for the contribution, and so forth. In the future, we will compare several methods and discuss their effects.

5. RELATED WORK

As the famous ontology building tools, we compare with Protégé, Hozo, and KiWi. Protégé [2] is the most famous ontology editor mainly used by ontology researchers for more than ten years. It has several advanced features like ontology inference. Also, it opens the plug-in specification, and now has more than 70 plug-ins in its official site such as data import from RDB, term extractions by text mining, etc. Hozo [8] is a tool which has a unique feature to handle Role

concept. It also has a distributed development environment that keeps data consistency by checking the difference of ontologies edited by multiple users.

However, those are the heavyweight ontology tools mainly for the ontology experts. In the future, we will refer to them for the ontology visualization plug-in and the mechanism to keep the consistency of data, and so forth. However, KiWi (Knowledge in A Wiki) [9] focused on the lightweight ontology, and the extended Wiki by the semantic web technology. It enables the user to edit its content through the same interface as the Wiki, and so the user can register the instances and properties without any special knowledge of ontology and the tool. It is a different method than our editor, whereas for the same purposes like an easy introduction for the non-expert, the improvement of usability, and the collaborative work. In the future, we would also like to incorporate the Wiki-based change history and difference management mechanism.

Moreover, there have been several researches to extract the information from textual documents on the Web, that are combining Natural Language Processing (NLP) mechanisms and semantic resources like ontologies. Our extraction mechanism is similar to NELL [10] with the features like “Ontology-driven”, “Macro-reading” of the NELL. This means that the input is a large text collection and the desired output is a large collection of facts, using “Machine learning methods”. However, instead of the Coupled Pattern Learner in NELL, we used a morphological analysis and a dependency parsing. Moreover, clustering of the values using LCS and the PageRank are also our own methods. Also, a key strategic difference is a target domain of the extraction. The NELL is targeting the world, so that the granularity of the properties is large and the number of the properties is limited. For example, “agricultural product growing in state or province” is a barely fine-grain property in the NELL, whereas only 10 instances have this property. Also, the number of the properties is only 5% of the total extracted terms. However, by restricting the domain of interest, it is possible for our mechanism to construct the set of the co-occurrence strings with the predefined property name. This simple heuristic effectively select candidates for the property values, and then raise the accuracy of the extraction and keep the variety of the properties.

Furthermore, NERD (Named Entity Recognition and Disambiguation) framework [11] has proposed an RDF/OWL-based NLP Interchange Format (NIF) and an API to unify various tools for a qualitative comparison. LODifier [12] has recently proposed a translation of textual information in its entirety into a structural RDF in open-domain scenarios with no predefined schema. In the future, we would like to use the techniques for further improvement.

6. CONCLUSIONS

This paper presented the development of a web-based ontology editor for the non-expert. First, we raised three problems for such editors and attempted to solve them with five approaches. Especially we described the mechanism of <*property, value*> extraction from the Web in order to reduce the workload of the term registration, and confirm the feasibility of showing the binary choice by the evaluation. We continue to offer the ontology tool for the developers and users of the consumer services incorporating a diverse range of their opinions.

REFERENCES

[1] B. Faltings, V. Schickel-Zuber, 2007. Oss: A semantic similarity function based on hierarchical ontologies. Proceedings of 20th Int. Joint Conf. on Artifical Intelligence (IJCAI 2007), pp.551-556.

[2] The Protégé Ontology Editor and Knowledge Acquisition System, <http://protege.stanford.edu/>.

[3] OWL, Web Ontology Language, <http://www.w3.org/TR/owl-features/>.

[4] Adobe Flex, < http://www.adobe.com/products/flex/>.

[5] Jena - A Semantic Web Framework for Java, <http://jena.sourceforge.net/>.

[6] S. Brin: “Extracting patterns and relations from the world wide web”, Proc. of WebDB Workshop at 6th International Conference on Extended Database Technology, 1998.

[7] T. Kawamura, S. Nagano, M. Inaba, Y. Mizoguchi: Mobile Service for Reputation Extraction from Weblogs - Public Experiment and Evaluation, Proc. of Twenty-Second Conference on Artificial Intelligence (AAAI), 2007.

[8] Hozo, <http://www.hozo.jp/ >.

[9] KiWi - Knowledge In A Wiki, <http://www.kiwi-project.eu/>.

[10] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E.R. Hruschka Jr. and T.M. Mitchell: Toward an Architecture for Never-Ending Language Learning, Proc. of Conference on Artificial Intelligence (AAAI), 2010.

[11] G. Rizzo, R. Troncy, S. Hellmann, M. Bruemmer: NERD meets NIF: Lifting NLP Extraction Results to the Linked Data Cloud, Proc. of 5th Workshop on Linked Data on the Web (LDOW), 2012.

[12] I. Augenstein, S. Pado, and S. Rudolph: LODifier: Generating Linked Data from Unstructured Text, Proc. of 9th Extended Semantic Web Conference (ESWC), 2012.

Authors

Takahiro Kawamura is a Senior Research Scientist at the Corporate Research and Development Center, Toshiba Corp., and also an Associate Professor at the Graduate School of Information Systems, the University of Electro-Communications, Japan.

I Shin received a M.S. degree at the Graduate School of Information Systems, the University of Electro-Communications, Japan and joined NTT DATA Corporation in 2010.

Akihiko Ohsuga is a Professor at the Graduate School of Information Systems, the University of Electro-Communications. He is currently the Chair of the IEEE Computer Society Japan Chapter.