Tag Ontology Automatic Building for Semantic Searching of Services: a Case Study on Mashup Services

Weifeng Pan
School of Computer Science and Information Engineering
Zhejiang Gongshang University
Hangzhou 310018, PRC
Email: panweifeng1982@gmail.com
Shan Li
Tencent Technology (Shenzhen) Company Limited
Shenzhen 518000, PRC
Email: scorpiolishan@gmail.com

Abstract— **The explosion of services like web services, APIs, Mashups, etc., makes how to find the right one you need a tough problem. Tags, as a kind of metadata have been widely used to annotate services. In this paper, we propose to use an ontology automatically built from tags to improve the performance of service searching. We use the famous Mashup directory, Programmable.com, to illustrate our approach: First, all metadata especially tags of mashups and APIs at Programmable.com are crawled and preprocessed by the suffix stripping algorithm to lower the noise. Second, a Mashup-Mashup Network (MMN) is constructed to represent the Mashups and their relationships. And the community detection technique in complex network theory is introduced to mine the community structures (the potential domains of these Mashups) in MMN. Based on the Mashup communities, we further group the tags into corresponding domains. Finally the tag ontology is built with reference to WordNet, and embedded in our own developed software service registry and repository (S2R2), to improve the performance of service searching. Comprehensive experiments are also conduct to testify the effectiveness of the proposed approach.**

Index Terms— **ontology, folksonomy, services, tags, community detection, WordNet, complex networks**

I. Introduction

With the development of Web 2.0 technology, the number of services on the Internet is increasing. Thus, how to retrieve the desired services has become a major issue. Compared with those keyword-based systems, the ontology-based system using the ontology as a conceptual model at semantic level [1], is viewed as an promising way to capture the semantic meaning of the query string and improve the searching experiences. However, ontology is mainly built by domain experts manually, making the ontology building a high cost and long development cycle process [2]. The ontology is also difficult to evolve, which can not meeting the explosion of service resources. Thus, how to build ontology automatically, and further to support service searching is an urgent problem.

Labeling is a kind of folksonomy techniques [3], and has been widely used in service registration systems to annotate services. The tags they used contain much information about the services, especially their functions, providing a new source of semantics. Indeed, tags mainly play two roles: (1) organization of the services, and (2) a bridge that connects users and services. But to the best of our knowledge, the work on the use of the roles services take (especially the second role) to improve service searching has never been reported. This is the motivation of the current work.

In this paper, we propose to build a tag ontology automatically to support semantic searching of services. A famous Mashup directory, Programmable.com [4], is used as an illustration to show how to perform our approach. We crawl the Mashups and APIs services in Programmable.com, and use the suffix stripping algorithm to preprocess the tags. Then based on the data collected, a Mashup-Mashup Network (MMN) is constructed to represent the micro-topology of Mashups and their relationships. The community detection technique in complex network theory is introduced to mine the community structures in MMN. Based on the Mashup communities, we calculate a metric to quantify the weights of each tag to these communities, grouping the tags into corresponding communities. The tag communities are the potential domain of these tags. With reference to the WordNet [5], a lexical database for English, we can extract the hierarchical structure of the tags in each community. And an ontology can also be built. We also embedded the tag ontology into our own developed software service registry and repository (S2R2) [6] to improve the performance of service searching.

The rest of this paper is organized as follows. Section II describes our approach in detail, with focus on the formal definitions of Mashup-Mashup network, the algorithm to

Corresponding Author: Weifeng Pan
This work was supported by the Zhejiang Provincial Natural Science Foundation of China under Grant No. LQ12F02011 and the Science Research Project of Zhejiang Gongshang University under Grant No. 1130KU112021.

doi:10.4304/jcp.7.12.2979-2986

detect the communities and the procedure to build the tag ontology. Section III performs an experiment to show the effectiveness and efficiency of the proposed approach. Section IV contains a brief summary of the related work. And we conclude this paper in section V.

II. The approach

In the following sections, we will first detail the data we crawled from the Programmableweb.com. And then we preprocess these data using the suffix stripping algorithm to lower the noise. After that these data are represented as networks. And a community detection algorithm will be applied to find the communities. With reference to the WordNet, a tag ontology will be built. Figure 1 gives a short overview of the workflow of the proposed approach. In the following subsections we will detail the key steps one by one.

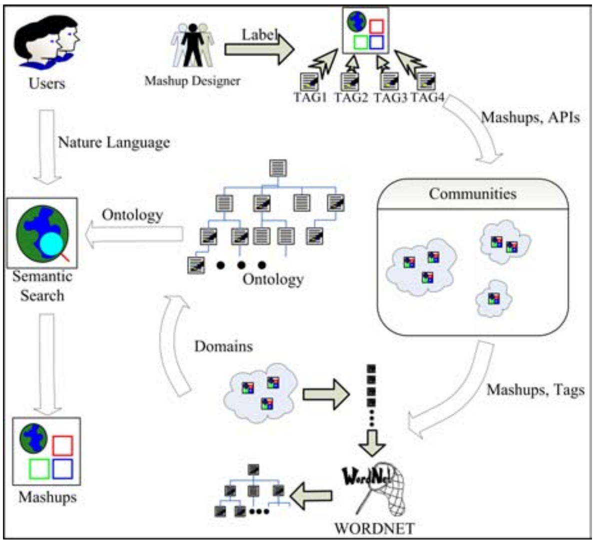


Figure 1. The workflow of the proposed approach.

A. Data Source

With the increase of the number of services, many online service registration sprang up such as Yahoo pipes [7], Xignite splice [8], and ProgrammableWeb.com for Mashup and API services. It provides a third party platform for service designer to publish their services, and also for users to discover the needed services. Compared with other platforms mentioned above, ProgrammableWeb.com takes a great different line that it does not require the Mashups and APIs registered should be developed using the tools it provides.

Programmableweb.com is such a community that allows you to publish the Mashups you designed by simply filling in the URL, description, date, API used, and most importantly, the tags annotated to this Mashup. It also provides the detail information about a list of APIs that you can used to design your Mashups. The API information includes description, URL, summary, category, data Format, protocol, provider and tags.

We crawl the metadata of Mashups, including tags, APIs used, name, URL, etc. from Programmableweb.com from the date of its establishment 2005 to March 26, 2010 and store them into database. These data constitute the data source of the current work.

B. Tag Preprocessing

Since the tags are randomly chose by the Mashup developers from their own vocabularies, tags with the same meaning will always have different representations in Programmable.com, and even have a wrong spelling. For example, api, Api, $APIs$ are labeled to different Mashups, but they indeed share the same meaning API. So we should preprocess these tags to lower the noise in the data.

There are a lot of approaches that can be used to preprocess the tags. In the current work, we choose to use the suffix stripping algorithm (SSA) presented by M.F. Porter in [9]. Since it is open source and can be easily combined into our S2R2 platform.

SSA starts from a state that any word, or part of word, has the form as:

$$[C](VC)^m[V], \tag{1}$$

where C represents a list of consonants with length greater than 0, V represents a list of vowels with length greater than 0, the square brackets denote arbitrary presence of their contents, and m represents the repeated times. For example, the tag $connections$ can be described in the form $C(VC)^3$.

The rules for removing a suffix will be described as:

$$(condition)S1 \rightarrow S2 \tag{2}$$

This means that if the tag ends with the suffix $S1$ and satisfies the condition in the parenthesis, the suffix $S1$ will be replaced by $S2$. Some notations used in the condition parts can be defined as: $*S$ denotes any tag which ends with S (and similarly for the other letters); $*v*$ denotes any tag that contains a vowel; $*d$ denotes any tag that ends with a double consonant (e.g. -TT, -SS); and $*o$ denotes any tag that ends cvc, where the second c is not W, X or Y (e.g. -WIL, -HOP).

The suffix stripping algorithm used in the current work is shown in Figure 2. For more details, please refer to [9].

There are some tags, which are combined by two words (such tags are also realistic searching keywords), e.g., the tag $microblog$ is combined by the tag $micro$ and the tag $blog$. In this paper, we will not preprocess these tags rather than delete them, and they will also be added into the ontology when the approach terminates.

C. Network Construction

In our work, we will represent the Mashups, APIs and their relationships by two kinds of networks, API-Mashup

Input: Output:	All tags Tags pretreated		
	a	b	c
Step 1	SSES → SS IES → I SS → SS S →	(m>0) EED → EE (*v*) ED → (*v*) ING →	(*v*) Y → I
Step 2	(m>0) ATIONAL → ATE, (m>0) ANCL → ANCE, (m>0) ALLI → AL, (m>0) OUSLI → OUS, (m>0) ATOR → ATE, (m>0) FULNESS → FUL, (m>0) IVITI → IVE,	(m>0) TIONAL → TION, (m>0) IZER → IZE, (m>0) ENTLI → ENT, (m>0) IZATION → IZE, (m>0) ALISM → AL, (m>0) OUSNESS → OUS, (m>0) BILITI → BLE	(m>0) ENCL → ENCE (m>0) ABLI → ABLE (m>0) ELI → E (m>0) ATION → ATE (m>0) IVENESS → IVE (m>0) ALITI → AL
Step 3	(m>0) ICATE → IC, (m>0) ICITI → IC, (m>0) NESS →	(m>0) ATIVE → , (m>0) ICAL → IC,	(m>0) ALIZE → AL (m>0) FUL →
Step 4	(m>1) AL → , (m>1) ER → , (m>1) IBLE → , (m>1) MENT → , (m>1) and (*S or *T) ION → (m>1) OU → , (m>1) ITI → ,	(m>1) ANCE → , (m>1) IC → , (m>1) ANT → , (m>1) ENT → , (m>1) ISM → , (m>1) OUS → ,	(m>1) ENCE → (m>1) ABLE → (m>1) EMENT → (m>1) IZE → (m>1) ATE → (m>1) IVE →
Step 5	a: (m>1) E → (m=1 and not *o) E →	b: (m>1 and *d and *L) →	

Figure 2. Suffix stripping algorithm [9].

affiliation network and Mashup-Mashup network. And the former is the basis to construct the latter. First we will give the formal definitions of the two kinds of networks.

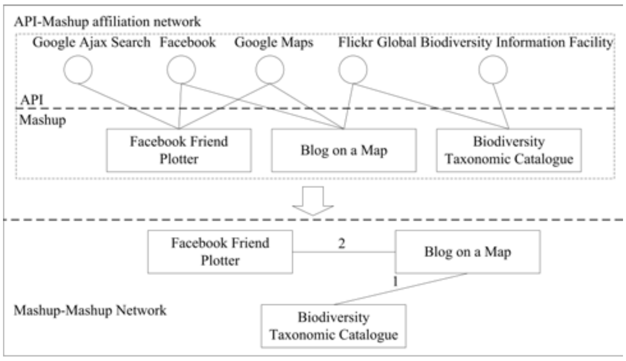


Figure 3. An illustration of AMAN (API-Mashup affiliation network) and its corresponding MMN.

Definition 1: API-Mashup Affiliation Network

The API-Mashup affiliation network (AMAN) is a bipartite network, $\mathrm{AMAN} = (N_m, N_a, D)$, that explicits the relationship between Mashups and APIs. The bipartite network consists of two sets of nodes N_m (Mashups) and N_a (APIs). Only links between nodes of unlike set are allowed, that is $D = \{m_i, a_j\}$ where $m_i \in N_m$ and $a_j \in N_a$. Note that the adjacency matrix ψ_{ij} for the bipartite network encodes the connections between Mashups and APIs:

$$\psi_{ij} = \begin{cases} 1 & \{a_i, m_j\} \in D \\ 0 & otherwise \end{cases} \tag{3}$$

Definition 2: Mashup-Mashup Network

The Mashup-Mashup network (MMN) is a weighted network. The nodes in MMN represent Mashups, edges represent the similarity relationship of the two Mashups linked by the edges, and the weight on each edge represents their similarity. Therefore MMN can be described as:

$$\mathrm{MMN} = (N_m, E_m), \tag{4}$$

where N_m is the set of Mashups and E_m is the set of edges denoting all functional similarity relationships among Mashups. The relationships between Mashup nodes can be recovered by means of one-mode projection of the bipartite network [10]. The adjacency matrix ψ^m for the network MMN is related to the adjacency matrix ψ by

$$\psi_{ij}^m = \sum_k \psi_{ik}\psi_{jk}, \tag{5}$$

In this paper, the similarity between two Mashups is decided and calculated by the number of APIs shared by the two Mashups in AMAN, i.e., if Mashup i and Mashup j share APIs in AMAN, there will be an edge linking them together, and the number of APIs shared is the similarity value on the edge. See Figure 3 for an illustration. Since Mashup $Facebook\ Friend\ Plotter$ and $Blog\ on\ a\ Map$ share APIs $Facebook$ and $Google\ Maps$, so there will be an edge between the nodes denoting the two Mashups, and the number of shared APIs, 2, is the similarity value on the edge. So does that of

Mashups $Blog\ on\ a\ Map$ and $Biodiversity\ Taxonomic\ Catalogue$. And Mashups $Facebook\ Friend\ Plotter$ and $Biodiversity\ Taxonomic\ Catalogue$ share no API, so there is no edge in the corresponding MMN.

D. Mashup Community Detection in MMN

Services in a registration usually belong to many different sub-domains. In order to detect the sub-domains existing in MMN, the community detection techniques in complex network theory will be applied. And the communities detected, in fact, are the sub-domains that the Mashups belong to.

Community structure, the gathering of nodes into groups such that there is a higher density of edges within groups than between them, is one of the network features that has been emphasized in recent work. The problem of community detection has been well studied and a lot of algorithms have been proposed such as Kernighan-Lin algorithm [11] and spectral portioning [12-13].

A popular method now widely used relies on the optimization of a similarity measure, which is a quality index for a partition of a network into communities. And in this paper, we will also use such a similarity measure to evaluate the community structures in MMN. Considering the large scale of services, here we introduce a modified fast algorithm (MFA) to mine the communities in MMN.

1) Similarity Measure: There are many different quality functions to test whether a particular division is meaningful, such as MQ introduced by Mancoridis et al. [14], EVM function of Tucker et al. [15] and modularity Q devised by Newman and Girvan [16]. In this paper, we use the Q metric simply for its popularity. But the original Q is proposed to detect the communities in unweighted networks. In order to make it suitable to weighted networks, we propose its weighted version which is read as:

$$Qw = \sum_i (we_{ii} - {wa_i}^2), \tag{6}$$

where Qw is the similarity of a particular division, we_{ii} is the fraction of the total weight of the edges that connect two nodes within community i, while wa_i is the fraction of the total weight of the edges that have at least one endpoint within community i.

But the Qw is devised to evaluate the effectiveness of the whole division. How can we use it to evaluate the similarity between communities? To address this problem, we borrow the basic idea from Newman's fast algorithm. In this paper, we will use ΔQw to measure the similarly between two communities. Because if two communities with the maximum similarity are divided into two different communities, there will be a decrease in modularity Qw, otherwise there will be an increase. The value of ΔQw denotes the similarity between two communities. So to search the communities with the most similarity means to find the largest ΔQw. This strategy to accelerate the speed of the algorithm is very similar to that proposed in [16], [17], and [18].

The change in Qw upon joining communities i and j is given by:

$$\Delta Qw = \begin{cases} we_{ij} + we_{ji} - 2wa_i wa_j & i, j \ is \\ & connected \\ 0 & otherwise \end{cases} \tag{7}$$

In this paper, we iteratively search for the changes ΔQw resulted from the amalgamation of each pair of communities, and choose the largest them, until there is only one community left.

2) A Modified Fast Algorithm: The modified fast algorithm (MFA) used to detect the communities in MMN is shown in Algorithm 1. It has been successfully applied in other situations such as class refactoring and service classification, showing its great performance. For details, please refer to [17] and [18].

Algorithm 1 A Modified Fast Algorithm

Input:
MMN
Output:
Qw and communities detected

1: Assign each node in MMN as a community
2: Calculate Qw according to (6), and calculate ΔQw_{ij} for pairs of communities according to (7), and store the ΔQw_{ij} in ΔQw matrix
3: **while** number of communities > 1 **do**
4: Select the largest ΔQw_{ij} from ΔQw matrix
5: Merge the corresponding communities i and j, update ΔQw matrix and increase Qw by ΔQw_{ij}
6: **end while**

E. Ontology extraction

In this section, we mainly focus on the procedures used to build the tag ontology.

1) Tag Weight Measure: After the Mashup communities are detected, the tags associated to them are also grouped. However, the same tags may appear in different communities (domains), and this is inconsistent with the concept in ontology. Thus, we should assign each tag to only one community. Here, we propose a metric tag weight, denoted as W_T to fulfill this task and construct the tag communities. The tag weight of a tag in a specific community is defined as:

$$W_T = \begin{cases} \sum_i W_i & if\ W_T = \max(W_{T1}, W_{T2}, ..., W_{Tn}) \\ 0 & else \end{cases} \tag{8}$$

where W_{Tj} is the weight of a tag corresponding to community j. And we define W_i=1 if the ith Mashup in a community is labeled by this tag, otherwise, W_i=0. W_{Tj} is the tag weight of the tag in community j. And the final W_T is the maximum value of W_{Tj}.

When performing the experiment, we found there are some tags, which have poor cognitive meanings to the services using these tags. So these tags are not helpful for improving the performance of service searching. Here, we ignore these tags by only keeping tags whose weights are larger than a specific threshold. And the threshold is determined by repeated tries.

2) NOUN Hierarchy Organization in the WordNet: We will build a tag ontology, relationships in which are hypernymy and hyponymy. To build the tag ontology objectively and automatically, we will refer to the organization structure of nouns in the WordNet.

WordNet is an online lexical database based on psycholinguistic principles. Nouns in the WordNet are organized into sets of synonyms (also called synset). And there are various semantic relations between these synonym sets. A synset represents a concept, and contains a set of words, each of which has a sense. Different senses of words are in different synsets. The semantic relations between hyponymy (sub-name) and its inverse, hypernymy (super-name) are transitive relations between synsets. E.g. , the word cat in the WordNet belongs to the synset cat, $true\ cat$ - ($feline\ mammal\ usually\ having\ thick\ soft\ fur\ and\ no\ ability\ to\ roar$; $domestic\ cats$; $wildcats$) and its hypernym synset is $feline$, $felid$ - ($any\ of\ various\ lithe\text{-}bodied\ roundheadedv\ fissiped\ mammalsv\ many\ with\ retractile\ claws$). Such a semantic organization can be fully exploited to build the tag ontology.

Ontology captures the domain concepts and their relations. Corresponding, WordNet has the structure of synsets and hypernymy/hyponymy relations. Therefore, we can build the ontology for tags based on the WordNet, which will be detailed in the next section.

3) Ontology Extraction Algorithm: In order to build the tag ontology, the relationships between tags and concepts in the WordNet should be extracted first. By attaching the tags to a suitable nodes in the WordNet, the tag ontology can then be built. The tag-concept relationship extraction algorithm is shown in Algorithm 2 with the computational complexity being $O(AB^2)$ (A is the number of tags and B is the number of words in the WordNet).

Step 5 shows that if T contains a tag that is the hypernym of the select one, there is no need to lookup these tags again. Such a step can improve the performance of the algorithm. Figure 4 gives an example of the structure extracted from the WordNet. Each node in the figure means a synset in the WordNet, and the nodes with a circle means there is a tag in this synset.

III. EXPERIMENT

In this section, we will use the data crawled to show the results and validate the effectiveness of our approach.

Figure 5(a) gives the statistics of the data collected. Figure 5(b) examines the growth of the number of Mashups over the time, which showing a nearly linear tendency. It is interesting.

In the process of data crawling, we find some errors made in ProgrammableWeb.com. Some APIs not being

Algorithm 2 Tag-Concept Relationship Extraction Algorithm

Input:
tag set T, WordNet
Output:
tag ontology

1: **while** T is not NULL **do**
2: Select a tag from T
3: Obtain the synset involved the tag by looking up the WordNet
4: Get all of the ancestors of the synset in the WordNet
5: If T contains the words belonging to the ancestors synsets, delete them from T
6: Treat every synset as a class in the ontology, and the words belonging to the synset as the properties of the class
7: **while** The words have direct hypernym synsets **do**
8: Treat hypernym synsets as the super-class
9: Let the words be the hypernym synsets
10: **end while**
11: **end while**

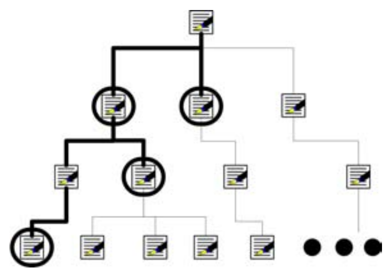
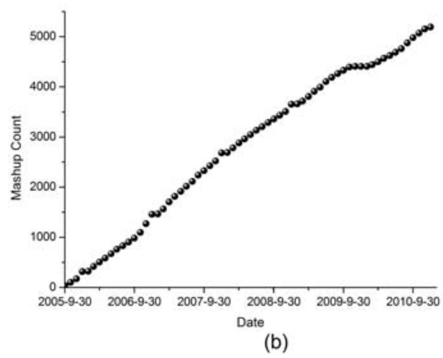


Figure 4. Illustration of the structure extracted from the WordNet.

	Name (X)	Count (Y)
1	Mashup	5191
2	Mashup tag	1466
3	Mashup provider	2365
4	API	748

(a)



(b)

Figure 5. The statistics of the data crawled from Programmableweb.com.

listed in the API directory also have been used by some Mashups, which does not conform to our instinct. So we add these APIs manually to our data set. And for those Mashups that have been repeatedly registered

TABLE I.
TAG STATISTICS

	Original Tags	*Preprocessed Tags*
Number of Tags	1,466	1,212
Examples	animals; animation; anime; api; Apis; attractions; band; bands; bike; bikes; biking; clothes; clothing	anim; anim; anim; api; api; attract; band; band; bike; bike; bike; cloth; cloth

in ProgrammableWeb.com (i.e., Mashups with the same metadata have been listed more than one time in Mashup directory), we only store one copy of them in our data set, i.e., the same Mashup will only be stored once. So the Mashup count shown in Figure 5(a) is smaller than that shown in Programmableweb.com.

Table I shows the tag statistics before and after pre-processing, and an illustration of the preprocessed tags compared with the original tags.

Since several API services are shared by large number of Mashups, the resulting MMN is too dense for direct visualization (average degree $< k > \simeq 1,010$). Figure 6 only shows a small part of the MMN we built, together with the Mashup communities MFA detected (nodes with the same color). Indeed, in our experiment, there are total 15 communities detected in MMN. But other 11 communities so small in size (Mashup count) that can be ignored statistically. Meanwhile, according to the tag weight, every tag can also be grouped into a specific community. It is obviously can be seen from Figure 6 that the first community (blue nodes) is concerned about traffic, the second (yellow nodes) is mainly about the entertainment online, e.g., mp3, demo, video, mail, audio, etc., the third community (green nodes) is mainly about the maps (Mashups in this category almost all use the API $Google\ Maps$), and the fourth is concerned about communication (phones and mobiles).

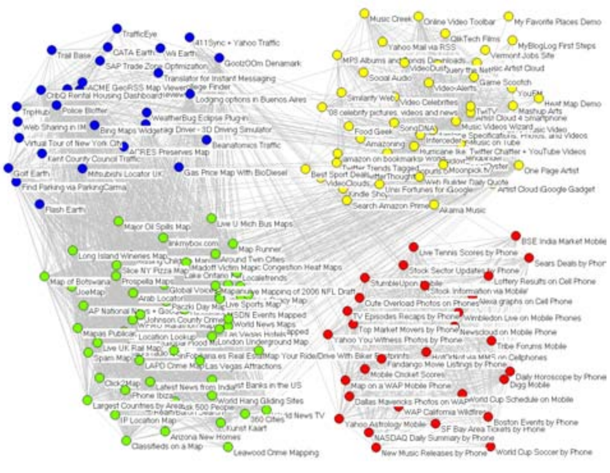


Figure 6. Illustration of a small part of MMN.

By referring to the WordNet, the tag ontology can be built. Since there is a large number of concepts in the tag ontology, the direct visualization of the ontology is not clear. Here we only show the traffic part as an illustration (see Figure 7). And the whole ontology is available for download from [19]. Such a tag ontology is the definition of a community name in the root of the tags hierarchy. As children concepts, all tags of this community will be organized hierarchically according to their organization structures in the WordNet.

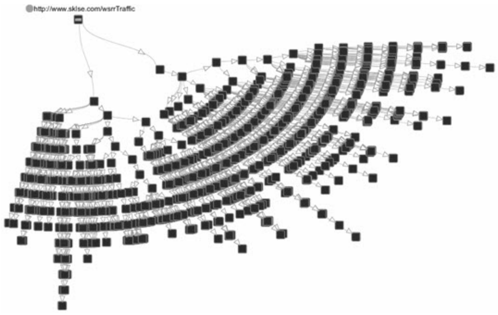


Figure 7. The traffic part of the tag ontology.

Finally, we embedded this tag ontology in S2R2. The snapshot of the S2R2 can be found in Figure 8, where the top left is the homepage of S2R2, the top right is the searching page, the button left is the results when using key-words $gift$ and buy, and the button right is the details of the Mashup $Friendcup$. The process is realized by firstly extracting the semantic information from the words users used for searching, and then matching the information with the concepts in the tag ontology. Finally, Mashup services related with the concepts is recommended to users.

In this experiment, the effectiveness of our approach is evaluated by recall and precision [20], two metrics that have been widely used in pattern recognition and information retrieval. Here we borrow these two metrics from information retrieval, and adapt them to fit in with the service searching problem.

In service searching, precision is the fraction of retrieved services that are relevant to the search. It reads:

$$\text{precision} = \frac{|\{\text{retrieved relevant services}\}|}{|\text{retrieved services}|} \tag{9}$$

Recall is the fraction of the services that are relevant to the query that are successfully retrieved. It is defined as

$$\text{recall} = \frac{|\{\text{retrieved relevant services}\}|}{|\text{relevant services}|} \tag{10}$$

The experiment is carried out in one of my classes. There are 31 students in the class. All of them are the first time to use S2R2. They are asked to use any word or phrase just as that they used in Google and Yahoo search engine, to search for services in our system. S2R2 will return two versions of results: one is obtained by using the tag ontology, and the other one is obtained by using the traditional keyword-based searching. And we store the precision and recall values of all search requests.

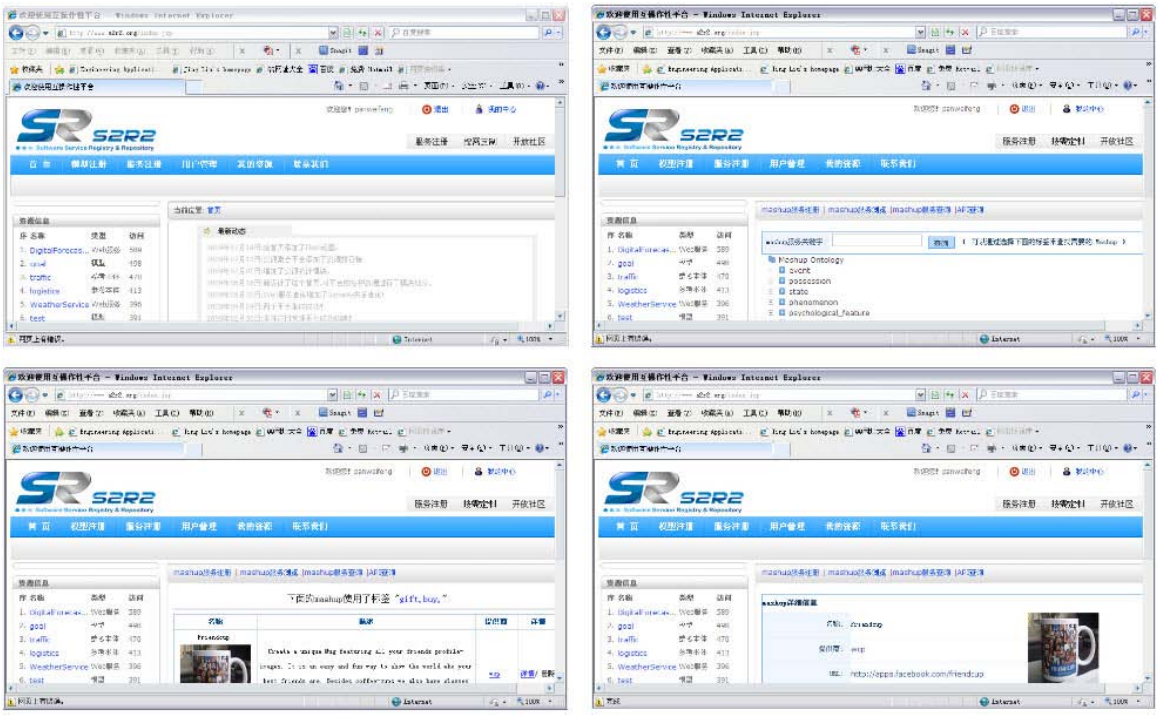


Figure 8. Some snapshots of S2R2.

The experiment lasts about 15 minutes. There are 1,246 times of searching requests. We find that under the same recall, the precision value (averaged over the 1,246 searching requests) of our approach is obviously larger than that using traditional key-word based searching (see Figure 9). It because our approach, which combined keyword-based searching and ontology technologies, is more likely to understand the meaning hidden in the words that users use. Further, the tag ontology we built can grow up with the growing resource in the website, simply by rerunning the entire proposed approach.

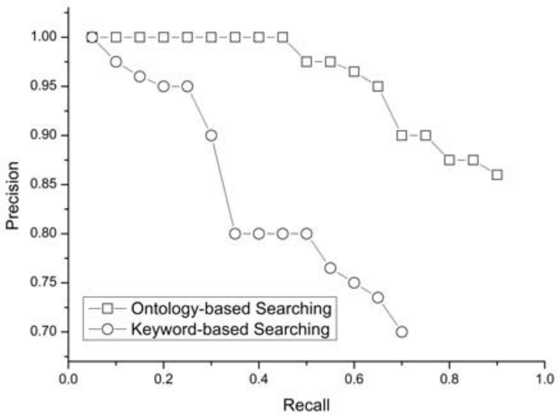


Figure 9. Recall vs. Precision.

IV. Related work

Many techniques such as boolean search [21], metadata [22], natural language ontology [23], etc., have been proposed to improve the performance of resource searching. But, for reasons of space, we only detail here some research work from the perspective of ontology-based searching.

B. Chang et al. presented an ontology-based educational information search service, which focus on the aspect of education information [24]. In [25], V. Torres et al. proposed a semantic query approach in the form of a navigational ontology, which focus on web applications. In [26], D. Chen et al. presented a system that using SPARQL to query the global ontology, and then searching the information in the database. They uses a semantic mapping table to constitute the mapping results between the information and ontology. However, if the information in the database belongs to many different domains, the global ontology will be very large for searching.

Though this is not the first work on ontology-based searching, we cover a different angle. Different from these approaches that using a domain ontology, our approach focus on the tags labeled by users. We build the tag ontology automatically and objectively, and then use the ontology to improve the performance of service searching. In short, our approach is based on folksonomy techniques, not the mastership of experts. Of course, ontology can be used in other domains such as mechanical design of learning contents [27] and workflow customization [28].

V. Conclusion

In the era of service explosion, how to search service efficiently and exactly becomes a growing problem. In this paper, we focus on the building of a tag ontology to enhance the semantic searching of services, and final to tackle this problem. It is a folksonomy way, and can be easily extended to other applications. Throughout the current work, we use Programmableweb.com, a Mashup and API directory, as an running example to illustrate our approach, including the procedure to preprocess tags, detect communities and build the tag ontology. Experimental results shown our approach got better performance when compared with the traditional keyword-based approach. Further, our tag ontology has a property that can grow up with the growing resource. The only work should be done is to rerun the entire proposed approach.

In the future research, we will focus on the following areas: 1) mine more semantic information from these tags; 2) optimize the tag ontology to make semantic search more efficiently; and 3) add the *hot* tag created by users to the tag ontology.

All the data used in this paper are available for download from [19].

REFERENCES

[1] Y. Fei, W. Tong. Research on construction and encoding method of virtual query ontology tree. In Proceedings of the 9th International Conference on Hybrid Intelligent Systems, Shenyang, China, Aug. 12-14, 2009, pp. 350-354.

[2] M. Morneau, Guy W. Mineau. Employing a domain specific ontology to perform semantic search. In Proceedings of the 16th International Conference on Conceptual Structure, Toulouse, France, July 7-11, 2008, pp. 242-254.

[3] C.V. Damme, M. Hepp, K. Simpaes. Folksontology: an integrated approach for turning folksonomies into ontologies. In Proceedings of the ESWC Workshop Bridging the Gap between Semantic Web and Web 2.0, Innsbruck, Austria, June 3-7, 2007, pp. 57-70.

[4] ProgrammableWeb.com. Available at: http://www.programmableweb.com.

[5] A. Miller. WordNet: an on-line lexical resource. International Journal of Lexicography, 1990, 3(4): 235-244.

[6] S2R2. Available at: http://www.s2r2.org.

[7] Yahoo pipes. Available at: http://pipes.yahoo.com/pipes.

[8] Xignite splice. Available at: http://splice.xignite.com.

[9] M.F. Porter. An algorithm for suffix stripping. Program, 1980, 14: 130-137.

[10] R. Diestel. Graph theory, grad. texts in math. Springer, 2005.

[11] B.W. Kernighan, S. Lin. An efficient heuristic procedure for partitioning graphs. Bell System Technical Journal, 1970, 49(1): 291-307.

[12] M.E.J. Newman. Finding community structure in networks using the eigenvectors of Matrices. Physical Review E, 2006, 74(3): 036104.

[13] L. Angelini, S. Boccaletti, D. Marinazzo, M. Pellicoro, S. Stramaglia. Fast identification of network modules by optimization of ratio association. arXiv:cond-mat/0610182v2, 2006.

[14] S. Mancoridis,B.S. Mitchell, C. Rorres, et al. Using automatic clustering to produce high-level system organizations of source code. In Proceedings of 6th International Workshop on Program Comprehension, Ischia, Italy, June 24-26, 1998, pp. 45C53.

[15] A. Tucker, S. Swift, X. Liu. Grouping multivariate time series via correlation. IEEE Transaction on Systems, Man, and Cybernetics. Part B. Cybernetics, 2001, 31(2): 235C245.

[16] M.E.J. Newman. Fast algorithm for detecting community structure in networks. Physical Review, 2004, 69: 066133.

[17] W.F. Pan, B. Li, Y.T. Ma, J. Liu, Y.Y. Qin. Class structure refactoring of object-oriented softwares using community detection in dependency networks. Frontiers of Computer Science in China, 2009, 3(3): 396-404.

[18] W.F. Pan, B. Li, B. Shao, P. He. Service classification and recommendation based on software networks. Chinese Journal of Computers, 34(12): 2355-2369.

[19] Data. Available at: http://www.whucn.com/wfpan.htm

[20] J. Makhoul, F. Kubala, R. Schwartz, R. Weischedel. Performance measures for information extraction. In Proceedings of DARPA Broadcast News Workshop, Herndon, VA, February, 1999.

[21] C.P. Paice. Soft evaluation of boolean search queries in information retrieval systems. Information Technology: Research and Development, 1984, 3(1): 33-42.

[22] B. Hughes, A. Kamat. A metadata search engine for digital language archives. D-Lib Magazine, 2005, 11(2): 6.

[23] A. Bernstein, E. Kaufmann. GINO-a guided input natural language ontology editor. In Proceedings of the 5th International Semantic Web Conference, Athens, GA, USA, November 5-9, 2006, pp. 144-157.

[24] B. Chang, D. Ham, D. Moon, Y. Choi, J. Cha. Educational information search service using ontology. In proceedings of the 7th IEEE International Conference on Advanced Learning Technologies, Niigata, Japan, July 18-20, 2007, pp. 414-415.

[25] V. Torres, J. Fons, V. Pelechano, O. Pastor. Navigational modeling and the semantic web: an ontology based approach". In Proceedings of the WebMedia LA-Web 2004 Joint Conference, Preto-SP, Brazil, Oct. 12-15, 2004, pp. 94-96.

[26] D. Chen, L. Li, R. Wang. An ontology-based system for semantic query over heterogeneous databases. World-Congress on Software Engineering, Xiamen, China, May 19-21, 2009, pp. 50-53.

[27] J. Zhou, K. Watanuki. Skill ontology for mechanical design of learning contents. Journal of Software, 2012, 7(1): 61-67.

[28] C.H. Liu, J.J.J. Chen. Using ontology-based BDI agent to dynamically customize workflow and bind sementic web service. Journal of Software, 2012, 7(4): 884-894.

Weifeng Pan received his Ph.D. degree in computer science from Wuhan University (WHU), China, in 2011.

He is now a Lecturer in School of Computer Science and Information Engineering (SCIE) at Zhejiang Gongshang University (ZJGSU), China. He is also a member of China Computer Federation (CCF), and Association for Computing Machinery (ACM). His current research interests include software engineering, service computing, complex networks, and intelligent computation.

Shan Li received his M.S. degree in computer science from Wuhan University (WHU), China, in 2011.

He is currently a Software Engineer at Tencent Technology (Shenzhen) company limited, China. His research interests include service computing and cloud computing.