Interface-Compatibility-Based Semantic Web API Discovery

Shang-Pin Ma[1], Hsuan-Ju Lin[1], Ying-Jen Chen[1], Wen-Tin Lee[2], and Hsi-Min Chen[3]
[1]Department of Computer Science and Engineering, National Taiwan Ocean University, Keelung, Taiwan
[2]Department of Software Engineering, National Kaohsiung Normal University, Kaohsiung, Taiwan
[3]Department of Information Engineering and Computer Science, Feng Chia University, Taiching, Taiwan
E-mail: albert@ntou.edu.tw, mis101bird@gmail.com, jane15751@gmail.com, wtlee@nknu.edu.tw, hsiminc@fcu.edu.tw

Abstract—**In recent years, the Web API (application programmable interface) gains more and more attractions. and the REST (REpresentational State Transfer) is widely accepted as the mainstream of Web API provision style. However, current Web API search engines provide merely either keyword search or tag-based search. It is not easy for users to find suitable Web APIs since the current search engines consider neither the semantics of Web APIs nor the characteristics of Web APIs, such as interface compatibility. Therefore, in this paper, we propose a RESTful service discovery approach, called Interface-Compatibility-based Semantic Service Search (ICS^3). ICS^3 expands terms in service documents based on DBpedia and WordNet and filters out inappropriate services for the user query by two steps: 1) calculating semantic similarities between candidate services and the user query and 2) analyzing the degrees of interface compatibility between candidate services and the user query by performing the Hungarian Algorithm. Integrating interface analysis with semantic term expansion can bring two benefits: 1) increase the possibility of matching semantically equivalent or similar services for the user query; and 2) reduce the time complexity for the Hungarian-based interface matching by filtering out candidate service with low similarity scores. The experimental results show that ICS^3 can achieve better accuracy than the traditional IR-Based approach.**

Keywords—service discovery, RESTful service, term expansion interface compatibility

I. INTRODUCTION

In recent years, with the rapid development of web technology, mobile computing and social network, the Web API (application programmable interface) gains more and more attractions. Lots of well-known companies, such as Google, Facebook, Netflix, eBay, LinkedIn, Foursquare, Instagram, and many others, published their Web APIs, and acquired explicit and implicit benefits. Developers could build their applications by integrating Web APIs to avoid spending considerable effort on non-core functionalities. According to the analysis by the well-known ProgrammbleWeb [1] website, the REST (REpresentational State Transfer [2]) is the mainstream of Web API provision style [3]. An enormous number of public RESTful services can be found on the internet. For example, there are more than 7,000 RESTful services published in the ProgrammbleWeb. Along with the booming development of RESTful services, the REST becomes the most popular software component model nowadays.

Although service discovery is not a new research area, very few existing efforts applied their methods on RESTful services. Meanwhile, current Web API search engines, such as ProgrammbleWeb [1], apis.io [4], and mashape [5], provide merely either keyword search or tag-based search. It is not easy for users to find suitable RESTful services [6] since current search engines consider neither the semantics of services nor the characteristics of services, such as input parameters, output data, and interface compatibility. Therefore, how to effectively and efficiently retrieve appropriate RESTful services for users is definitely an important problem to be solved.

Based on the above analysis, in this paper, we propose a Web API discovery approach, called Interface-Compatibility-based Semantic Service Search (ICS^3). ICS^3 borrowed two concepts in past researches on service discovery and composition: interface matching and semantic term expansion [7]. Interface matching, usually applied in software component retrieval [8] and service composition [9], must rely on exact matching or advanced subsumption matching [10] for I/O (input/output) elements. It is usually impractical to apply traditional interface matching because 1) exact matching may hinder the retrieval of semantically equivalent or similar services for the user query; and 2) subsumption matching can only limitedly improve the matching by relaxing the matching of I/O elements based on "IS-A" relationships. Meanwhile, exhausted matching of I/O elements between the user query and all candidate services may have high time complexity. To address the above issues, ICS^3 semantically expands terms in service documents based on DBpedia [11] and WordNet [6, 12], and filters out appropriate services for the user query by two steps: 1) calculating semantic similarity scores between services and the user query based on VSM (vector space model) and 2) analyzing the degrees of interface compatibility between services and the query by performing the Hungarian Algorithm [13] to ensure that the retrieved services can be compatible with the specified interface in the user query. Integrating Hungarian-based interface matching with semantic term expansion can bring two benefits: 1) increase the possibility of matching semantically equivalent or similar services for the user request; and 2) reduce the time complexity for the interface matching by filtering out candidate service with low similarity scores.

The remainder of this paper is organized as follows. Background work and related work is presented in Section 2. Section 3 outlines the details of the proposed approach. Section 4 presents the experiments used for the evaluation of the proposed approach. Section 5 summarizes the benefits and features of the proposed approach.

II. BACKGROUND WORK AND RELATED WORK

In this sections, we introduce important background work for the proposed approach and describe representative related works, including efforts on SOAP service discovery and approaches for RESTful service discovery.

A. Ontology-Related Techniques

Ontology can be used to describe the relationships between resources or concepts, which can be divided into Class and Property. Class has components such as superclass and subclass. Property has components such as range, domain, and subProperty.

1) *DBpedia*

DBpedia [11] is the core of linked data nowadays. Tim Berners-Lee, the director of the World Wide Web Consortium (W3C), proposed linked data in 2006. The goal of linked data is to build a data network, whose data are interlinked and become more useful through semantic queries. In addition, it shares information in a way that can be read automatically by computers. DBpedia is a semantic web containing extracted data from Wikipedia, and provides each data with one URL for search and storing. Besides, DBpedia allows users to search for RDF (Resource Description Framework) in DBpedia by SPARQL (SPARQL Protocol and RDF Query Language). DBpedia includes almost 700 Classes, and 3,000 Properties, which is the biggest ontology dataset in the world. In addition, it also includes over 4 million resources. In this paper, we use DBpedia ontology and example of resources to annotate input and output parameters of service documents, and use SPARQL by Jena[1] plugin to search DBpedia ontology and example of resources.

2) *WordNet*

WordNet [6, 12] is a large lexical database of English. In WordNet, nouns, verbs, adjectives and adverbs are grouped into sets of synsets, each expressing a distinct concept. Synsets are interlinked by meaning of conceptually-semantic and lexical relations. The linked relations include antonym, hypernym, hyponym, member holonym, etc. The relation between hypernym and hyponym is similar to IS-A relation in OO programming. The newest version 3.0 of WordNet is released, and it includes more than 117,000 synsets. In this paper, we use the term relations such as synonym, hypernym, and hyponym of WordNet for term expansion.

3) *Hungarian Algorithm*

Hungarian Algorithm [13], proposed by Harold Kuhn in 1955, is a combinatorial optimization algorithm that solves the assignment problem in polynomial time. The reason why the algorithm is called Hungarian is that the algorithm is on the basis of works from Hungarian mathematicians, Dénes Kőnig and Jenő Egerváry. The algorithm is easier to describe if we formulate the problem using a bipartite graph. Bipartite graph is a graph whose vertices can be partitioned into two sets A and B and no edge has both endpoints in the same set. Every possible edge that could connect vertices in different sets is part of the graph. Hungarian Algorithm can find the best one-on-one mappings of A and B in this problem. In Figure 1, we can see vertices A (circle points) connect vertices B (square points) with edges, and each edge has the cost of connection. We use Hungarian Algorithm to find the mappings with the minimal cost (seven), {A0-B2, A1-B0, A2-B1, A3-B3}, in this example. In this paper, we use the Hungarian Algorithm on the mapping analysis of input and output parameters, to check the interface compatibility between the query and the service.

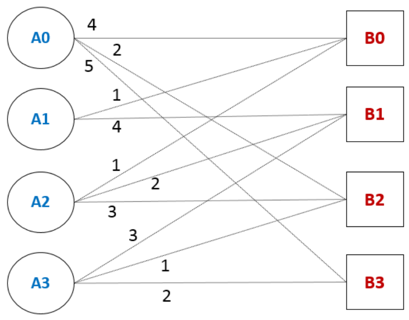

Figure 1. The example of bipartite graph

The Hungarian Algorithm has an $O(n^3)$ time complexity, which causes high processing time. Thus, we filter out inappropriate services by VSM-based service matching before conducting interface-compatibility-based search to reduce the response time.

B. SOAP Service Discovery Methods

The existing SOAP service discovery methods can be classified to two categories. One is text-based search, calculating the similarity between a query document and service documents via IR (information retrieval) methods. Stroulia and Wang [14] used TF-IDF (term frequency/inverse document frequency) and WordNet techniques to calculate the similarity between two WSDL documents by considering data format, service operation, and service content. Dong et al. proposed Woogle [15] search engine to calculate the similarity between queries and service operations according to term relationships built by a clustering algorithm. Hao et al. presented an IR-based service discovery and sorting method [16] which takes the service relevance and the service importance as indicators to enhance the precision of service discovery. Plebani et al. proposed a service search method named URBE (UDDI Registry By Example) [17], which is based on UDDI (Universal Description, Discovery, and Integration). URBE analyzed structures of WSDL documents and terms used in WSDL to measure the similarity between multiple Web service interfaces. URBE also utilized WordNet and domain ontology to realize a semantic-oriented variant for enhancing precision. Another branch of service discovery uses ontology to describe, search, and compose services semantically. Verma et al. [18] used

[1] https://jena.apache.org/

annotation method (SAWSDL) to add semantics in WSDL to improve the accuracy of searching for web services. Martin et al. utilized OWL-S [19] to describe the web service capability semantically and store semantic information in UDDI [20]. Jiang et al. [9] considered QoS and interface compatibility on service composition and applied Hungarian Algorithm to achieve optimum combination of I/O mapping between component services. Note that the semantics was not considered in this approach because service composition needs exact interface matching. Conversely, ICS^3 integrated Hungarian-based interface matching with semantic term expansion to increase the possibility of retrieving candidate services which are capable of fulfilling the user query.

2.4 RESTful Service Discovery Methods

In the RESTful service discovery field, Khorasgani et al. proposed a Semantic Flow Matching (SFM) method [21] to search for candidate RESTful services. The method analyzed input parameters of RESTful services, which are described by WSDL, and used input parameters to build a semantic-oriented variant web with WordNet. Users could find the services with the same function by the web. Wu et al. proposed a service matching method based on the Parameter Semantic Network model [22]. The method parses API documents and extracts three kinds of parameters: subject parameters, request parameters, and response parameters after pre-processing. Then it calculates similarities between APIs via term comparison, based on WordNet. In the end, it builds a similarity matrix by the similarities to train service data and accordingly to obtain better matching results. The above RESTful service discovery methods do not consider service interface compatibility, which is the focus of our approach.

III. INTERFACE-COMPATIBILITY-BASED SEMANTIC SERVICE SEARCH

In this chapter, we first describe how we perform the pre-processing of documents, introduce the proposed semantic RESTful service search, and finally explain the interface-compatibility-based semantic search.

A. Pre-processing of Documents

After a query document or a service document is delivered to ICS^3, ICS^3 will do preparatory tasks for the issued queries and the published service documents in order to perform index building and semantic term extension. The process includes following procedures.

1) *Tokenization*

We use the IR tool, Lucene Analysis API[2] to chop a service description up into pieces, and to remove stop words. In this paper, we collect general terms for RESTful services to build our stop word lists, in order to enhance the search precision.

2) *Stemming*

Many words are derivations from the same stem and we can consider they belong to the same concept (e.g., organize and organization). These derivations are generated through appended affixes (prefixes, infixes, and/or suffixes) and we can use the well-known stemming algorithm, Porter Stemming Algorithm [23], to strip these suffixes from a derived word. After obtaining stems of terms (e.g., organ), all the related words can be identified by matching their stems. In this paper, we do stemming on terms by Porter Stemming Algorithm, and store the stems in database.

a) *Semantic Term Expansion*

It is common that both the query terms issued by the user and the terms included in the service descriptions are not exactly matched the regulated terms, causing the low precision of service matching. Therefore, in this paper, ICS^3 performs term expansion for each service document. When a parameter in a service document is annotated as an ontology class or a property, the annotated term will be expanded based on WordNet and DBpedia; When a parameter in a service document is annotated as a resource, the term annotated with resource will be expanded merely on the basis of WordNet. The details of expansions are fully discussed in following sub-sections.

b) *DBpedia-Based Term Expansion*

In service documents, each input and output parameter is annotated by DBpedia through a new-defined property: mappingType. For example, in Facebook Graph Search Place API, the mappingType for the input parameter "name" is linked to "DBpedia:ontology/Place", and the mappingType of the output parameter is linked to "DBpedia:ontology/City". ICS^3 expands the annotated term of each parameter in service documents to generate an expansion bag. The expansion process includes the following steps:

- If mappingType is annotated with a DBpedia class, we use the class to include its three-level super classes and its direct sub classes to the bag.
- If mappingType is annotated with a DBpedia property, we find the class assigned to the range and the class assigned to the domain, and include the class as well as its three-level super classes and its direct sub classes to the bag.
- If mappingType is annotated with resource, we only do lexical expansion by WordNet.

In the past, there are many methods for calculating the similarity between two terms. In this paper, we use Edge Counting Method [24] to calculate the similarity between two terms, which takes path length between terms and depth of the subsume into consideration. The format is defined in Equation 1.

---

[2] https://lucene.apache.org/

$$sim(w1,w2) = f(l) * f(h) = e^{-\alpha l} * \frac{e^{-\beta h} - e^{\beta h}}{e^{-\beta h} + e^{\beta h}} \rightarrow [0..1]$$

Equation 1.

where *l* is the shortest path length between *w1* and *w2*, and *h* is the depth of the subsumer, which is derived by counting the levels from the subsumer to the top of the hierarchy. To avoid bias setting impacting the proposed approach, we followed suggestions in [24] to set α and β to 0.2 and 0.6 by default.

By the method, we can calculate the similarity between the original class and the expanded class, and the similarity is ranging from 0 to 1. For example, superclass "DBpedia:ontology/Settlement" is a subclass of "DBpedia:ontology/City" and the similarity *Sim(city, settlement)* is equal to 0.78.

Finally, only the terms whose similarities are larger than θ_{oe} will be classified into a set of expanded terms ExT. For example, we set θ_{oe} to 0.7, and the term "settlement" would be classified to ExT. In the further processing, the calculated similarity of the term becomes its weight.

c) *WordNet-Based Term Expansion*

In this paper, besides the proposed ontological expansion, we also use WordNet to do lexical expansion. In this stage, we obtain the synset of each term by WordNet, including synonyms, hypernyms and hyponyms. In WordNet, the synonym is on the same level as the original term; the hypernym is more abstract than the level of the original term; the hyponym is more concrete than the original term. We calculate the similarity between each term of synset (synonyms, hypernyms, hyponyms) and the original term. Similar to the DBpedia-based term expansion, only the terms whose similarities are larger than θ_{we} (currently we also set θ_{we} to 0.7) will be classified into a set of expanded terms ExT and be passed to the next stage. Besides, we calculate the TF (term frequency) for each term in ExT, and multiply its weight by TF to emphasize the importance of occurrence frequency of terms.

B. VSM-Based Semantic Service Search

After performing above steps, we obtain service semantic information by two stage of expansion and store them in database. When a query document is issued, ICS^3 firstly performs tokenization and term stemming for the document. Secondly, ICS^3 conducts term comparison between the set of extracted tokens for the query document and the set of expanded terms for each published RESTful service. Finally, ICS^3 calculates the similarity by VSM (vector space model), and then ranks the services according to the calculated similarities. The details of the similarity computation are described as follows in depth.

In this research, we divide service and request information into three sets of terms: descriptions, inputs and outputs, and each term in these sets contains its weight. We build the query vector and the service vector on the basis of these sets. A query vector $Q = \{dv_q, iv_q, ov_q\}$ includes three parts: query description vector dv_q, query input vector iv_q and query output vector $\underline{ov_q}$; a service vector, $S = \{dv_s, iv_s, ov_s\}$ also includes three parts: service description vector dv_s, service input vector iv_s and service output vector ov_s.

Then we calculate the similarity between the query vector and the service vector by VSM according to Equation 2.

$$SS(Q,S) = \frac{dv_q \cdot dv_s + iv_q \cdot iv_s + ov_q \cdot ov_s}{\sqrt{|dv_s|^2 + |iv_s|^2 + |ov_s|^2} \cdot \sqrt{|dv_i|^2 + |iv_i|^2 + |ov_i|^2}}$$

Equation 2.

After the similarity computation, candidate services whose similarities are larger than θ_{SS} (in this research, we assign it to 0.5 to filter out about half candidate services) will be passed to the next stage, interface compatibility analysis stage.

C. Interface-Compatibility-Based Service Search

In order to make sure that candidate services can be integrated with the developer's application smoothly, the interface of candidate services should match developer's expected service interface specified in the query document. In other words, candidate services should only require the prepared input parameters, without additional data, and generates expected output parameters, and even more data. For example, if a user plans to find services with *m* input parameters $I^r=\{i_1, i_2, i_3, \ldots, i_m\}$ and *n* output parameters $O^r=\{o_1, o_2, o_3, \ldots, o_n\}$, the number of input parameters of candidate services should be equal or lesser than n ($I^s \subseteq I^q$), and the number of output parameters of candidate services should be equal or more than *n* ($O^s \supseteq O^q$).

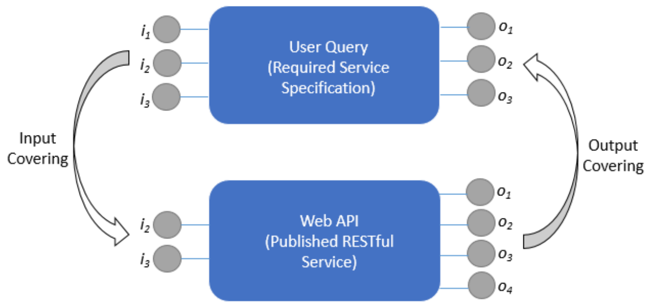

Figure 2. The concept of Input/Output Covering for interface compatibility analysis

The concept of the proposed Input/Output Covering (IOC) to analyze interface compatibility is shown in Figure 1. It illustrates input parameters of required service specification should cover published RESTful services, and the output parameters of published RESTful services should cover the required service specification. In addition, in order to solve the problem caused by approximate word or synonyms, which makes parameter mapping difficult, we design a method to achieve better parameter mappings.

As mentioned above, we need to find the optimum mappings for the analysis of interface compatibility. In this paper, in order to find the suitable combination of mappings between query parameters and service parameters, we calculate the sim_{map} for each query parameter and service parameter. The details of similarity computation are fully discussed below.

1) Determine the similarity between a query parameter and all extended terms for a service parameter.

2) Calculate the unadjusted similarity $sim\left(T_i^{qp}, ExT_{j_k}^{sp}\right)$ between a query parameter and an expanded tokens. Expanded tokens include the tokens extracted from the original service parameter.

3) Multiply the adjusted similarity $sim\left(T_i^{qp}, ExT_{j_k}^{sp}\right)$ by the weight of the expanded term $w(ExT_{j_k}^{sp})$ for the weight-similarity, $sim^w\left(T_i^{qp}, ExT_{j_k}^{sp}\right)$. The equation is as follows.

$$sim^w\left(T_i^{qp}, ExT_{j_k}^{sp}\right) = sim\left(T_i^{qp}, ExT_{j_k}^{sp}\right) \cdot w(ExT_{j_k}^{sp})$$

Equation 3.

4) Derive the maximum adjusted similarity between the query parameter and the expanded terms for a service parameter, and define it as the mapping similarity between the service parameter T_j^{sp} and the request parameter T_i^{qp} as follows.

$$sim^{map}\ (T_i^{qp}, T_j^{sp}) = Max\{sim^w\left(T_i^{qp}, ExT_{j_k}^{sp}\right), \forall k\}$$

Equation 4.

For example, an output parameter of the query is "name", and an output parameter of a service is "city" with expanded terms, including "urban_center" ($\text{weight} = 1$), "place" ($\text{weight} = 0.59$), "metropolis" ($\text{weight} = 1$), and "settlement" ($\text{weight} = 0.78$). We calculate the similarities of "name" and all expanded terms by Li's method and multiply them by their weights. We obtain the adjusted similarities *{ 0.0888 × 1, 0.0888×1, 0.4575×0.59, 0.2009×1, 0.2454×0.78, ... }* and then get the maximum adjusted similarity, which is between "name" and "city", such as $sim^{map}\ (name, city)$ = *Max {0.0888, 0.0888, 0.2699,0.2009, 0.1914,... } = 0.2699.*

Next, we continue to use the example above to explain the interface-compatibility calculation: the output parameters of the query are "name", "region" and "prefecture"; the output parameters of service are "name", "city", "coordinate" and "address". By the calculations of unadjusted similarities above, we can obtain mapping similarities for all parameter pairs like Table 1.

Table 1. The example of mapping parameter table

Candidate / Request	name	city	coordinate	address
name	1	0.27	0.24	0.63
region	0.23	0.55	0.36	0.45
prefecture	0.14	0.67	0.1	0.3

In addition to the calculations of the similarities between query and service parameters, we need to find the optimum combination of mappings. According to the interface compatibility concept we mentioned above, input parameters of a required service specification should cover published RESTful services, and output parameters of published RESTful services should cover the required service specification. Therefore, from the viewpoint of inputs, we find the optimum combination along the service input parameters by deriving the maximum sum of weights; conversely, from the viewpoint of outputs, we find the optimum combination along the query output parameters.

In this research, we apply the Hungarian Algorithm, as described in Section 2, to solve the issue of finding best mappings. Taking Table 1 as an example, the optimum combination of mappings calculated by Hungarian Algorithm is shown as Table 2. We obtain final mappings as {(name, name), (region, address), (prefecture, city)}.

Table 2. Optimum combination by Hungarian Algorithm

Candidate / Request	name	city	coordinate	address
name	**1**	0.27	0.24	0.63
region	0.23	0.55	0.36	**0.45**
prefecture	0.14	**0.67**	0.1	0.3

Based on the derived combination of mappings between the query and the service, we design a IOC scoring function (Equation 8.) to realize the concept of interface compatibility.

$$IOC(R,S) = \frac{\sum sim^{map}\ (T_i^{si}, T_j^{qi})}{m} \cdot w_{in} + \frac{\sum sim^{map}\ (T_i^{qo}, T_j^{so})}{n} \cdot w_{out}$$

Equation 5.

where T_i^{si} is input parameter of the service, and T_j^{qi} is input parameter of the query; T_i^{qo} is output parameter of the query, and T_j^{so} is output parameter of the service; m is the number of service parameters, and n is the number of query parameters. Besides, w_{in} and w_{out} are the weights to indicate the importance of inputs and outputs. Their sum is one and they are both assigned to 0.5 in this paper.

For example, in Table 2, we can obtain the combination of parameter mappings by the Hungarian Algorithm and calculate the output covering score:

($sim^{map}(name, name)$ + $sim^{map}(region, address)$ + $sim^{map}(prefecture, city)$)/3×0.5=

(1+0.45+0.67)/3×0.5=0.706=0.353

The calculation of the input covering score is similar to the output covering score. The difference between them is the "direction" of mappings: the mapping of inputs is from service input parameters to the query input parameters. If the calculated input covering score is 0.402, the final *IOC* score is 0.755 according to Equation 5.

After obtaining the *IOC* scores between the query and all services, ICS^3 filter out inappropriate services, whose IOC scores are not larger than the *IOC* threshold θ_{ioc} (currently we assigned it to 0.5), and treat the remaining services as candidate services.

Consequently, after the process of above two stages, ICS^3 collects *SS* scores and *IOC* scores of the candidate services and then calculates the final score ICS^3 score using the following equation.

$$ICS3(Q, S) = SS(Q, S) \cdot w_{ss} + IOC(Q, S) \cdot w_{ioc}$$

Equation 6.

where w_{ss} and w_{ioc} are the weights to indicate the importance of semantic search and interface-compatibility-based search. Their sum is one and they are both assigned to 0.5 in this paper.

Each candidate service S_c gets its degree of fitness with query Q by Equation 6. All the candidate services are ranked by $ICS3(Q, S_c)$ and send them back to users for further handing.

IV. EXPERIMENTAL EVALUATIONS

In order to demonstrate the feasibility of the proposed research, we implemented a prototype system, and use the prototype system to verify ICS^3, and compare it with full-text IR-based (information retrieval based) approach.

A. Experimental Configurations

The setup of the experiment was as follows:

1) Collect Web APIs on the internet, and submit service and query documents to the ICS^3 prototype.

We collected 200 RESTful services in various domains from ProgrammableWeb, Mashape, and APIs.io, developed corresponding service documents, including API descriptions, input and output parameters, and semantic mapping information, and stored them in the database. Meanwhile, we also wrote 14 representative query documents in diverse fields to be the testbed. A query document includes expected API title, input parameters, and output parameters.

2) Develop IR-based API search method.

We developed a search method on the basis of traditional IR (information retrieval) as the comparison target. Different from the ICS^3 method, the IR-based method does not divide query and service documents into three vectors, and does not perform term expansion based on DBpedia and WordNet expansion. The method includes the following steps:

a) Tokenization and stemming.

b) Calculating similarities between the query and all service documents by VSM (Vector Space Model).

c) Ranking services according to calculated similarities.

3) Set Parameters for ICS^3.

Currently we set the parameters as follows

$\theta_{oe} = 0.7$, θ_{we}=0.7, θ_{ss}=0.5, θ_{ioc}=0.5, w_{ss}=0.5, and w_{ioc}=0.5.

4) Apply various evaluation indicators to do verification.

We used multiple indications to demonstrate the performance of ICS^3, and compared ICS^3 with the intentionally-developed IR-Based API search method. The experimental indicators include Top-K precision (Equation. 3), Top-K recall (Equation. 4), R-precision (Equation. 5) and Response Time, i.e., the delay time (in milliseconds) between the moment the service query is issued and the moment the service search results are received.

$$Precision^K(Q_i) = \frac{|Rel(Q_i) \cap Rank^K(Q_i)|}{|Rank^K(Q_i)|}$$

Equation 7.

$$Recall^K(Q_i) = \frac{|Rel(Q_i) \cap Rank^K(Q_i)|}{|Rel(Q_i)|}$$

Equation 8.

$$Precision^R(Q_i) = \frac{|Rel(Q_i) \cap Rank^R(Q_i)|}{|Rank^R(Q_i)|}$$

Equation 9.

where, Q_i is the ith requests, and $Rank^K(Q_i)$ is the top-k services. $Rel(Q_i)$ is a set of relative services with Q, and R is the number of relavant services.

In a nutshell, we used 14 query documents to search for services by both ICS^3 and the IR-based method, and then gathered the search results with rankings to calculate the evaluation indicators: R-Precision, Top-K precision and Top-K recall (K is ranging from 1 to 10). Next, we comparted ICS^3 and the IR-based method by analyzing these indicators.

B. Experimental Results

The experiment results are shown in Figures 3 to 6. Average Top-K Precision is shown in Figure 3, Average Top-K Recall is Figure 4, R-Precision is Figure 5, and Response Time is Figure 6. The results show that the performance of ICS^3 is evidently better than the IR-based method from the viewpoints of either Precision or Recall. For the response time, ICS^3 indeed needs more processing time than the IR-based method. However, the average response of ICS^3, 0.28 seconds, is still in an acceptable level. Regarding the Average R-Precision, ICS^3 is obviously superior to the IR-based method (ICS^3: 0.714, IR-based method: 0.466). This indicator fully demonstrates that ICS^3 can precisely retrieve RESTful services that satisfy the users' needs.

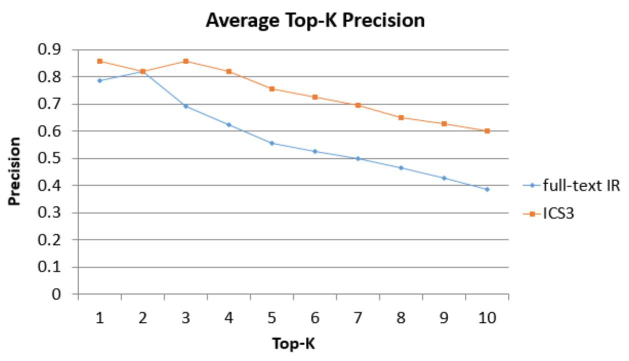

Figure 3. Average Top-K Precision

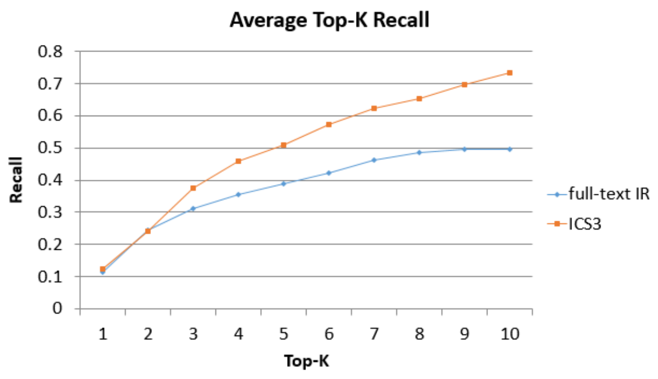

Figure 4. Average Top-K Recall

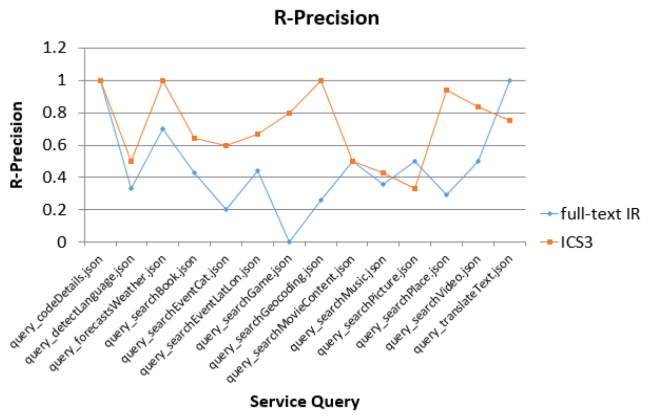

Figure 5. R-Precision

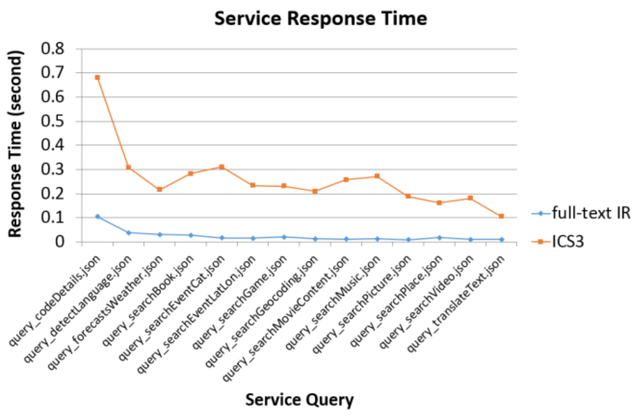

Figure 6. Response Time

V. CONCLUSION

In this paper, we propose a Web API discovery approach, called Interface-Compatibility-based Semantic Service Search (ICS^3), to precisely retrieve RESTful services. The main features of ICS^3 are fourfold: 1) ICS^3 can expand service documents by DBpedia and WordNet to increase the precision of service matching; 2) ICS^3 can calculate semantic similarities between services and the query to filter out irrelevant services; 3) ICS^3 can analyze the degrees of interface compatibility between services and the query and to filter out inappropriate services again; and 4) ICS^3 can determine the final service rankings based on the semantic similarities and the interface compatibility degrees. The experimental results show that ICS^3 can achieve better accuracy than the traditional IR-Based approach for the retrieval of Web APIs in REST style. Our future plan is applying the software testing techniques to further improve the accuracy of Web API retrieval.

References

[1] ProgrammableWeb. Available: http://www.programmableweb.com/

[2] R. T. Fielding, D. Software, and R. N. Taylor, "Principled Design of the Modern Web Architecture," ACM Transactions on Internet Technology, vol. 2, pp. 115--150, 2002.

[3] I. Gat and G. Succi. (2013) A Survey of the API Economy. Agile Product Management & Software Engineering Excellence.

[4] APIs.io. Available: http://apis.io/

[5] mashape. Available: https://www.mashape.com/

[6] G. A. Miller, "WordNet: a lexical database for English," Communications of the ACM, vol. 38, pp. 39-41, 1995.

[7] S.-P. Ma, C.-W. Lan, and C.-H. Li, "Contextual service discovery using term expansion and binding coverage analysis," Future Generation Computer Systems, vol. 48, pp. 73-81, 7// 2015.

[8] A. M. Zaremski and J. M. Wing, "Signature matching: a tool for using software libraries," ACM Trans. Softw. Eng. Methodol., vol. 4, pp. 146-170, 1995.

[9] C. Q. Jiang, W. Du, and C. C. Yi, "A Method of Web Service Composition Based on Bipartite Graph Optimal Matching and QoS," in Internet Technology and Applications, 2010 International Conference on, 2010, pp. 1-5.

[10] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara, "Semantic matching of web services capabilities," in The Semantic Web—ISWC 2002, ed: Springer, 2002, pp. 333-347.

[11] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, et al., "DBpedia - A crystallization point for the Web of Data," Web Semantics: Science, Services and Agents on the World Wide Web, vol. 7, pp. 154-165, 9// 2009.

[12] C. Fellbaum, WordNet: An Electronic Lexical Database, 1998.

[13] R. Jonker and T. Volgenant, "Improving the Hungarian assignment algorithm," Operations Research Letters, vol. 5, pp. 171-175, 1986.

[14] E. Stroulia and Y. Wang, "Structural and Semantic Matching for Assessing Web-service Similarity," Int. J. Cooperative Inf. Syst., vol. 14, pp. 407-438, 2005.

[15] X. Dong, J. Madhavan, and A. Halevy, "Mining structures for semantics," SIGKDD Explor. Newsl., vol. 6, pp. 53-60, 2004.

[16] Y. Hao, Y. Zhang, and J. Cao, "Web services discovery and rank: An information retrieval approach," Future Generation Computer Systems, vol. 26, pp. 1053-1062, 10// 2010.

[17] P. Plebani and B. Pernici, "URBE: Web Service Retrieval Based on Similarity Evaluation," IEEE Trans. on Knowl. and Data Eng., vol. 21, pp. 1629-1642, 2009.

[18] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Miller, "METEOR-S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services," Information Technology and Management, vol. 6, pp. 17-39, 2005/01/01 2005.

[19] D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, et al., "Bringing Semantics to Web Services: The OWL-S Approach," presented at the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004), San Diego, California, USA., 2004.

[20] D. Martin, M. Burstein, D. McDermott, S. McIlraith, M. Paolucci, K. Sycara, et al., "Bringing Semantics to Web Services with OWL-S," World Wide Web, vol. 10, pp. 243-277, 2007/09/01 2007.

[21] R. R. Khorasgani, E. Stroulia, and O. R. Zaiane, "Web service matching for RESTful web services," in 2011 13th IEEE International Symposium on Web Systems Evolution (WSE), 2011, pp. 115-124.

[22] Z. Wu, Z. Dou, and C. Song, "Web Service Matching for RESTful Web Services Based on Parameter Semantic Network," in 3rd International Conference on Computer Science and Service System, 2014.

[23] M. Porter. The Porter Stemming Algorithm. Available: http://www.tartarus.org/~martin/PorterStemmer/.

[24] Y. Li, Z. A. Bandar, and D. McLean, "An Approach for Measuring Semantic Similarity between Words Using Multiple Information Sources," IEEE Transactions on Knowledge and Data Engineering, vol. 15, pp. 871-882, 2003.