Web API Discovery Using Semantic Similarity and Hungarian Algorithm

Shang-Pin Ma[1], Hsuan-Ju Lin[1], Hsi-Min Chen[2], Ying-Jen Chen[1], Wen-Tin Lee[3]

[1] Department of Computer Science and Engineering, National Taiwan Ocean University, Taiwan
[2] Department of Information Engineering and Computer Science, Feng Chia University, Taiwan
[3] Department of Software Engineering, National Kaohsiung Normal University, Taiwan
albert@ntou.edu.tw, mis101bird@gmail.com, hsiminc@fcu.edu.tw, jane15751@gmail.com, wtlee@nknu.edu.tw

Abstract

Representational state transfer (REST) is the current design standard for Web application programmable interfaces (APIs). Unfortunately, existing Web API search engines allow for only keyword or tag-based searches. Furthermore, search engines do not take into account the semantics and the interface compatibility. This paper reports on a novel approach to RESTful service discovery, referred to as Interface-Compatibility-based Semantic Service Search (ICS^3). ICS^3 expands terms in service documents using DBpedia and WordNet, and then filters out services that are inapplicable to the user query, in two steps: (1) calculating semantic similarity between candidate services and the user query and (2) calculating the degree of interface compatibility between candidate services and the user query using the Hungarian algorithm. Experiment results demonstrate that ICS^3 outperforms other Web APIs search methods in terms of accuracy.

Keywords: Service discovery, RESTful service, Web API, Term expansion, Interface compatibility

1 Introduction

Rapid advances in web technology, mobile computing, and social networking have greatly enhanced the importance of Web API (application programmable interface) design. Google, Facebook, Netflix, eBay, LinkedIn, Foursquare, Instagram, and many others have published Web APIs, which can be used by developers to avoid the costs and effort spent associated with developing non-core functionalities. The well-known website, ProgrammbleWeb [1], has identified Representational State Transfer (REST) [2-3] as the standard for API design [4]. An enormous number of RESTful services [3] can be found on the internet, including the 7,000+ RESTful services published on ProgrammbleWeb. However, Web API search engines, such as ProgrammbleWeb [1], apis.io [5], and mashape [6], allow for only keyword or tag-based searches, which makes it very difficult for users to find suitable RESTful services. Furthermore, search engines do not take into account semantics or other characteristics of Web APIs, such as input parameters, output data, or interface compatibility.

In this paper, we propose a novel approach to the discovery of Web APIs, referred to as Interface-Compatibility-based Semantic Service Search (ICS^3). ICS^3 adopts two concepts from previous research on service discovery and composition: interface matching and semantic term expansion [7]. Interface is widely applied in the retrieval of software components [8]. Service composition [9] relies on exact matching or advanced subsumption matching [10] for I/O (input/output) elements. The application of conventional interface matching is generally impractical because exact matching tends to hinder the retrieval of semantically equivalent or similar services and subsumption matching is limited in its ability to improve performance by relaxing the matching of I/O elements based on "IS-A" relationships. The exhaustive matching of I/O elements between user queries and all candidate services imposes excessive computational complexity. To overcome these issues, ICS^3 uses DBpedia [11] and WordNet [12-13] to perform semantic term expansion in service documents. It also filters out services that are inapplicable to the user query by calculating semantic similarity scores between services and queries based on VSM (vector space model) and estimating interface compatibility using the Hungarian algorithm [14]. The integration of Hungarian-based interface matching with semantic term expansion increases the likelihood of finding services that are semantically equivalent or similar to the user request. It also reduces the time spent associated with interface matching by filtering out candidate service with low similarity scores.

The remainder of this paper is organised as follows. Section 2 presents an overview of the background and related works. Section 3 outlines the details of the

*Corresponding Author: Shang-Pin Ma; E-mail: albert@ntou.edu.tw
DOI: 10.3966/160792642018111906003

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.

2 Related Work

In this section, we outline related works, including the Hungarian algorithm, those involving SOAP service discovery, and various approaches to RESTful service discovery.

2.1 Hungarian Algorithm

The Hungarian algorithm [14], proposed by Harold Kuhn in 1955, is a combinatorial optimization algorithm based on work of Hungarian mathematicians, Dénes Kőnig and Jenő Egerváry. The algorithm is easier to describe if the problem of assignment in polynomial time is formulated using a bipartite graph, in which the vertices can be partitioned into two sets (*A* and *B*) with no edge, and with both endpoints in the same set. Thus, every possible edge that could connect the vertices in different sets forms part of the graph. The Hungarian algorithm can be used to find the best one-on-one mappings of *A* and *B*. As shown in Figure 1, the *A* vertices (circle points) connect to the *B* vertices (square points) with edges, each of which has the cost of connection. The Hungarian algorithm can be used to formulate mappings {*A0-B3, A1-B0, A2-B1, A3-B2*} and find the maximal outcome. In this paper, we employed the Hungarian algorithm to facilitate the mapping analysis of input and output parameters, in order to assess interface compatibility between queries and services.

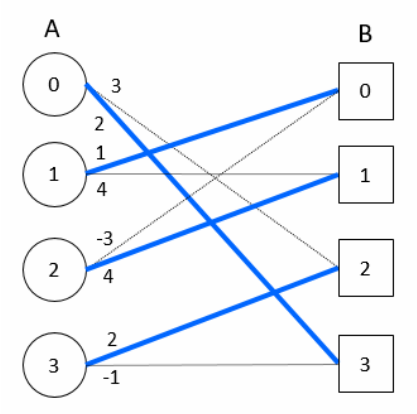


Figure 1. Example of bipartite graph matching

2.2 Service Discovery Methods

Existing service discovery methods include text-based searches, in which the similarity between a query document and service documents is calculated using IR (information retrieval) methods. Stroulia and Wang [15] used term frequency/inverse document frequency (TF-IDF) in conjunction with WordNet to calculate the similarity between two WSDL documents based on data format, service operation, and service content. Dong et al. developed the Woogle [16] search engine to calculate similarity between queries and service operations based on term relationships built using a clustering algorithm. Hao et al. presented an IR-based service discovery and sorting method [17] that uses relevance and importance of services as indicators to enhance the precision of service discovery. Plebani et al. proposed a service search method based on Universal Description, Discovery, and Integration (UDDI) [18], to analyze the structure and terms used in WSDL documents in order to measure similarity between multiple Web service interfaces. Their approach also uses WordNet and domain ontology to enhance precision.

Another approach to service discovery involves the use of ontology to describe, search for, and compose services based on semantics. Verma et al. [19] used an annotation method (SAWSDL) to include semantics in WSDL in order to improve accuracy in searching for web services. Martin et al. utilized OWL-S [20] to form semantic descriptions of web service capability and store semantic information in UDDI [21]. A third approach to service discovery is based on the quality of service (QoS). Hu et al. [22] proposed a service selection algorithm in which the preferences of multiple users are aggregated according to criteria derived from multiple providers, whereupon the most suitable service provider is selected based on voting results. J. Wu and Z. Wu [23] developed a suite of methods to assess similarity among Web services to facilitate matchmaking. They presented a conceptual model for the classification of Web service properties into Common Properties, Special Properties, Service Interface, and QoS as well as a variety of methods which could be used (together or individually) for the assessment of similarity to facilitate web service matchmaking. In [9], the method proposed in [23] were used in conjunction with QoS and interface compatibility in the application of the Hungarian algorithm and lexical similarity to optimize I/O mapping between component services. Note that the issue of semantics was not considered in this approach because service composition requires exact interface matching. ICS^3 integrates Hungarian-based interface matching with semantic term expansion to increase the likelihood of retrieving candidate services capable of fulfilling user requirements.

3 Interface-compatibility-based Semantic Service Search

In this chapter, we describe the pre-processing of documents, introduce the proposed semantic RESTful service search scheme, and outline the proposed interface-compatibility-based semantic search.

3.1 Pre-processing of Documents

After obtaining a query document or service document, ICS[3] conducts preparatory tasks to facilitate index building and semantic term extension. The process includes the following procedures.

3.1.1 Stemming and Tokenization

The IR tool, Lucene Analysis API[1], is used to chop a service description up into tokens and to parse a service description for the removal of stop words. We collected terms generally used for RESTful services in the construction of stop word lists, in order to enhance search precision.

Many words are derivations from the same stem, and therefore are subsumed under the same concept (e.g., organize and organization). The fact that these derivations are generated through appended affixes (prefixes, infixes, and/or suffixes) means that we can use the stemming algorithm, Porter Stemming Algorithm [24], to strip away the suffixes from the derived words. Obtaining the stems of the terms (e.g., organ) makes it possible to identify all of the related words simply by matching stems, which are then stored in a database.

3.1.2 Semantic Term Expansion

The discrepancies commonly encountered between the query terms submitted by users and the terms included in service descriptions can greatly undermine the accuracy of service matching. 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 property, the annotated term is expanded based on WordNet [12-13] and Dbpedia [11]. When a parameter in a service document is annotated as a resource, the associated term is expanded based only on WordNet. This expansion is detailed in following sub-sections.

DBpedia-based term expansion. In service documents, input and output parameters are annotated by Dbpedia through a newly defined property, mappingType. For example, in the Facebook Graph Search Place API, the mappingType for the input parameter "name" is linked to "DBpedia: ontology/Place", and the mappingType for 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 using a DBpedia class, then we use the class to include its three-level super classes and direct sub-classes in the bag.
- If mappingType is annotated using a DBpedia property, then we find the class assigned to the range and the class assigned to the domain, and include in the bag the two classes as well as its three-level super classes and direct sub-classes.

In this paper, we use the Edge Counting Method [25] to calculate similarity between two terms. This method takes into consideration the path length between terms as well as the depth of the subsumer. The formula to calculating simalrity is defined in Equation 1.

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

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 number of levels between the subsumer and the top of the hierarchy. To avoid the effects of bias, we followed the suggestions in [25] wherein α is set to 0.2 and β is set to 0.6 by default.

This approach method makes it possible to calculate the degree of similarity between the original class and the expanded class, over a range from 0 to 1. For example, superclass "DBpedia:ontology/Settlement" is a subclass of "DBpedia:ontology/City" with similarity *Sim(city, settlement)* of 0.78.

Finally, only the terms with similarity values exceeding θ_{oe} are classified within the set of expanded terms. For example, setting θ_{oe} to 0.7 would result in the inclusion of the term "settlement" within the set of expanded terms. In subsequent processing, the calculated similarity of the term becomes its weight.

WordNet-based term expansion. In addition to the proposed ontological expansion, we also employ WordNet for lexical expansion. WordNet [12-13] is a large lexical database of the English language, in which nouns, verbs, adjectives and adverbs are grouped into sets of synsets, each expressing a distinct concept. Using WordNet, we first obtain the synset of each term, including synonyms, hypernyms, and hyponyms. In WordNet, the synonym is on the same level as the original term, hypernyms are more abstract, and hyponyms are more concrete. We then calculate the similarity between each term in the synset (synonyms, hypernyms, hyponyms) and the original term. Similar to DBpedia-based term expansion, only the terms with similarity exceeding θ_{we} (currently we also set θ_{we} to 0.7) are classified within the set of expanded terms and passed to the next stage. We also calculate the term frequency (TF) for each expanded term, and multiply its weight by TF to emphasize the importance of frequency in the occurrence of terms.

1 https://lucene.apache.org/

Figure 2 demonstrates the result of term expansion. The mappingType property is assigned to the ontology class "dbpedia:ontology/City". As mentioned previously, if the mappingType is annotated as an ontology class, the annotated term is expanded based on WordNet as well as DBpedia. In DBpedia-based expansion, we expand three terms from super classes and two terms from direct subclasses. In WordNet-based expansion, we expand two synonyms from WordNet, and then calculate the weight for each expanded term.

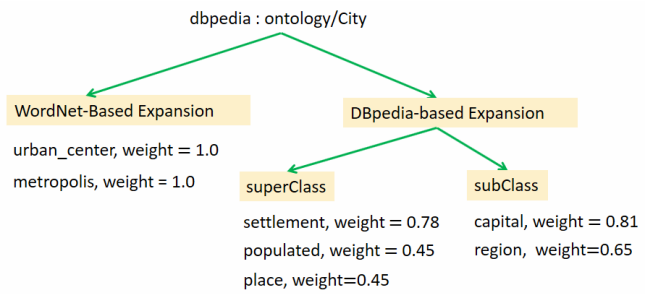


Figure 2. Example of term expansion

3.2 VSM-Based Semantic Service Search

We obtain semantic information pertaining to services in a two-stage process of expansion and then store it in a database. When ICS[3] first obtains a query document, it begins by performing tokenization and term stemming. It then conducts a comparison of terms 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 degree of similarity between the query and each published service using the well-known IR (information retrieval) method: VSM (vector space model), and ranks the services accordingly. The computation of similarity is detailed in the following.

In the proposed scheme, service and query information is divided into three sets of terms: descriptions (or titles), inputs, and outputs. Furthermore, each term in these sets is assigned a weight, which is the term frequency for the original term or is the product of the term frequency and the term similarity to the original term for an expanded term. The query vector and service vector are built based on these sets. A query vector $Q = \{dv_q, iv_q, ov_q\}$ includes a query description vector dv_q, query input vector iv_q, and query output vector ov_q. A service vector, $S = \{dv_s, iv_s, ov_s\}$ includes a service description vector dv_s, service input vector iv_s, and service output vector ov_s.

We calculate the degree of similarity between the query vector and the service vector using VSM, as follows (Equation 2):

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

Following the computation of similarity, candidate services with similarity exceeding θ_{ss} are passed on to the next stage to undergo analysis of interface compatibility. In this research, we set this to 0.5 to filter out approximately half of the candidate services based on the results of internal experiments.

3.3 Interface-compatibility-based Service Search

Ensuring the smooth integration of candidate services with developer applications requires that the interface of candidate services match the expectations specified in the query document. In other words, the invocation of candidate services requires only a subset of the prepared input data. The output of the service invocation generates data that contains the expected output parameters. For example, if a user plans to find services with m input parameters I^r and n output parameters O^r, then the number of input parameters (I^s) associated with a given candidate service should be equal to or less than m ($I^s \subseteq I^q$), and the number of output parameters (O^s) should be equal to or more than n ($O^s \supseteq O^q$).

We propose the IOC concept to ensure that (1) the input parameters of the required service specifications should cover published RESTful services, and (2) the output parameters of published RESTful services should cover the required service specifications. We developed a novel approach to parameter mapping to overcome the difficulties associated with word approximations and synonyms.

To obtain optimal mappings for the analysis of interface compatibility, we calculate the mapping for each query parameter and service parameter, as outlined in the following:

(1) Calculate unadjusted similarity $us(qt_i, st_{jk})$ between the ith query parameter and the kth expanded tokens for the jth original token. Expanded tokens include the tokens extracted from the original service parameter and semantically expanded terms.

(2) Multiply the unadjusted similarity $us(qt_i, st_{jk})$ by the weight of the expanded term $w(st_{jk})$ to compute the weight-similarity, $s(qt_i, st_{jk})$.

(3) Derive the maximum adjusted similarity between the query parameter and expanded terms for a service parameter, and define it as the mapping similarity between the service parameter qt_i and the query parameter st_j.

After calculating the similarity between query and service parameters, we must determine the optimum combination of mappings. As mentioned above, the input parameters of a required service specification should cover the published RESTful services, and the output parameters of published RESTful services should cover the required service specifications. Thus, from the perspective of inputs, we obtain the optimum

combination from the service input parameters by deriving the maximum sum of weights. From the perspective of outputs, we obtain the optimum combination from the query output parameters.

As described in Section 2, the Hungarian algorithm can be used to resolve the issue of finding the best mappings. Based on the resulting combination of mappings between the query and service, an IOC scoring function (Equation 3) is derived to fulfil interface compatibility, as follows:

$$IOC(R,S) = \frac{\sum ms(qit_i, sit_j)}{m} \cdot w_{in} + \frac{\sum ms(qot_i, sot_j)}{n} \cdot w_{out} \quad (3)$$

where qit_i is an input parameter of the service, and sit_j is an input parameter of the query; qot_i is an output parameter of the query, and sot_j is an output parameter of the service; m is the number of input parameters of the service, and n is the number of output parameters of the query. w_{in} and w_{out} are the weights indicating the importance of inputs and outputs with a sum of 1. In this paper, they are both assigned a value of 0.5.

The calculation of the input covering score is similar to that of the output covering score. The difference lies in the "direction" of mapping, wherein the mapping of inputs is from service input parameters to the query input parameters.

The above procedure is applied for the query and each published service. After obtaining the *IOC* scores between the query and all services, the ICS^3 filter out inappropriate services; i.e., those with IOC scores not exceeding the *IOC* threshold θ_{ioc} (set to 0.5) and treat the remaining services as candidate services.

ICS^3 then collects the *SS* scores and *IOC* scores of the candidate services to calculate the final score using the following equation:

$$ICS^3(Q,S) = SS(Q,S) \cdot w_{ss} + IOC(Q,S) \cdot w_{ioc} \quad (4)$$

where w_{ss} and w_{ioc} are the weights indicating the importance of the semantic search and interface-compatibility-based search with a sum of 1. They are both assigned a value of 0.5 in this paper.

Each candidate service S_c is assigned a degree of fitness with query Q using Equation 4. All candidate services undergo ranking by $ICS^3(Q, S_c)$ before being returned to the user.

4 Experimental Evaluations

To demonstrate the feasibility of the proposed ICS^3 scheme, we implemented a prototype system and compared the performance with that of the IR-based full-text method and semantic similarity of services (SimSS) method.

4.1 Experimental Configurations

The experiment was setup as follows:

Submission of service and query documents to the ICS^3 prototype. Although there are thousands of public RESTful APIs published in the Internet, it is hard to derive the service title/description, input parameters, and output parameters of all published services since there is no standardized way to describe the service interface. Meanwhile, the semantic mappings from the service description to the global ontology, such as DBpedia, are also lacking currently. To conduct experiments and perform evaluations, we collected a total of 200 RESTful services in various domains from ProgrammableWeb, Mashape, and APIs.io, and developed corresponding service documents, including API descriptions, input and output parameters, and semantic mapping information. We also wrote 14 representative query documents in a diversity of fields for use as a testbed. Query documents include only the expected API title, input parameters, and output parameters. Thus, Query documents are similar to simplified service documents.

Full-text API search method. We developed a search method based on the conventional IR approach (hereafter referred to as full-text), for use as a comparison target. Unlike 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. The IR method proceeds through the following steps:

(1) Perform tokenization and stemming.

(2) Calculate similarities between the query and all service documents using a VSM.

(3) Rank services according to calculated similarities.

SimSS search method. We established another search method, SimSS, based on [9] for use as a comparison target. This method takes into account the degree of semantic association, service quality, and efficiency in composition of web APIs. To enable a comparison with ICS^3, we made a number of modifications to the SimSS to allow single service discovery. The modified SimSS method proceeds through the following steps:

(1) Perform tokenization and stemming.

(2) Retrieve input and output parameters from query and API documents.

(3) Calculate mapping similarities (SimCC) between I/O parameter pairs (Equation 5).

$$simIn(Qi, Sj) = SimOut(Qi, Sj) = \frac{\alpha}{\alpha + d} \quad (5)$$

where d is the path length between terms and α is an adjusted value, set to 0.6.

(4) Use Hungarian algorithm to identify the best one-on-one mapping and calculate SimSS scores using the following equation (Equation 6).

$$SimSS(Q,S) = \frac{1}{n}\sum SimIn(Qi,Sj)\cdot w_{in} + \frac{1}{m}\sum SimOut(Qi,Sj)\cdot w_{out} \quad (6)$$

where w_{in} and w_{out} are 0.5, n is the number of service inputs, and m is the number of query outputs.

(5) Rank services according to SimSS scores.

Establishment of parameters for ICS^3. Parameter settings were based on the results of trial-and-error experiments, as follows: $\theta_{oe}=0.7, \theta_{we}=0.7$, $\theta_{ss}=0.5$, $\theta_{ioc}=0.5$, $w_{ss}=0.5$, and $w_{ioc}=0.5$.

Application of evaluation indicators for verification. Two indicators were used to evaluate the performance of ICS^3, the full-text method, and the SimSS method. The indicators included Top-K precision (Equation 7) and Top-K recall (Equation 8).

$$Precision^K(Q_i) = \frac{|Rel(Q_i)\cap Rank^K(Q_i)|}{|Rank^K(Q_i)|} \quad (7)$$

$$Recall^K(Q_i) = \frac{|Rel(Q_i)\cap Rank^K(Q_i)|}{|Rel(Q_i)|} \quad (8)$$

where, Q_i is the *i*th query, and $Rank^K(Q_i)$ is the Top-K retrieved services. $Rel(Q_i)$ is a set of relevant services with Q_i.

Top-K precision refers to the fraction of retrieved Top-K services that are relevant to the user and Top-K recall indicates the fraction of Top-K services that are relevant to the retrieved queries. These two indicators make it possible to determine the effectiveness of the full-text method, SimSS, and ICS^3.

We used 14 representative query documents in the search for services. We then gathered the search results and their rankings to calculate Top-K precision and Top-K recall where K ranges 1 to 10. The indicators were then used to evaluate the performance of ICS^3 and compare it with that of the full-text and SimSS methods.

4.2 Experiment Results

Average Top-K Precision is presented in Figure 3, and Average Top-K Recall is presented in Figure 4. These results illustrate how ICS^3 clearly outperforms the full-text and SimSS methods and clearly demonstrates the ability of ICS^3 to retrieve RESTful services capable of satisfying user requirements.

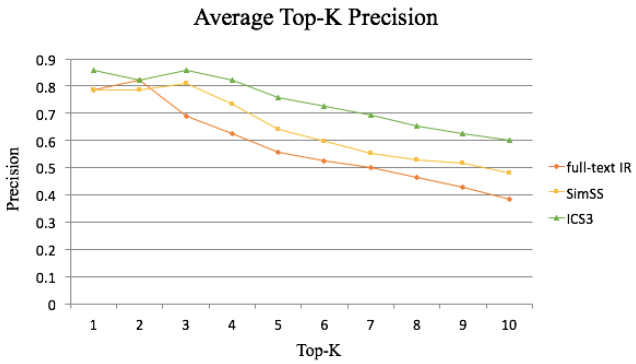


Figure 3. Average Top-K Precision

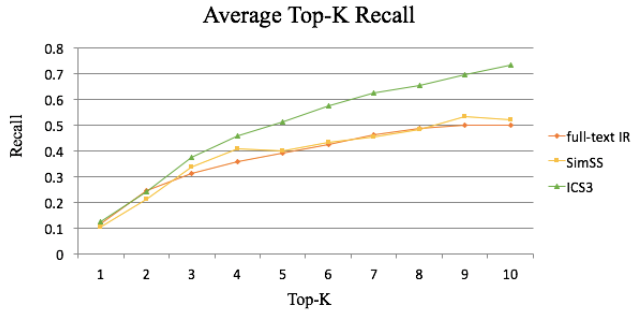


Figure 4. Average Top-K Recall

From the perspective of the Precision, although all three methods can accurately retrieve Top-1 service and Top-2 services, ICS^3 continues to keep the high Precision (0.6) even for the Top-10 retrieved services. From the perspective of the Recall, ICS^3 achieves 0.7+ Recall for the Top-10 retrieved services, whereas other two methods merely yield about 0.5 Recall.

5 Conclusion

This paper reports on a novel approach to Web API discovery, referred to as Interface-Compatibility-based Semantic Service Search (ICS^3), for the retrieval of RESTful services according to user-defined criteria. The main features of ICS^3 are four-fold: (1) ICS^3 expands service documents using DBpedia and WordNet to increase the precision of service matching; (2) ICS^3 calculates semantic similarities between services and queries to filter out irrelevant services; (3) ICS^3 analyses the compatibility of service and query interfaces to filter out irrelevant services; and (4) ICS^3 identifies the final service rankings based on the semantic similarities and the degree of interface compatibility. Experiment results demonstrate that ICS^3 achieves accuracy superior to that of the IR-Based approach and SimSS method. Note that although the proposed ICS^3 approach is used in the domain of RESTful services currently, it can be also applied to traditional SOAP services or other service models.

The next stage of development will involve the application of software testing techniques to further improve the accuracy of Web API retrieval.

Acknowledgments

This research was sponsored by Ministry of Science and Technology in Taiwan under the grant MOST 105-2221-E-019-054-MY3.

References

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

[2] R. T. Fielding, R. N. Taylor, Principled Design of the Modern Web Architecture, *ACM Transactions on Internet Technology*, Vol. 2, No. 2, pp. 115-150, May, 2002.

[3] J.-M. Gil, J. Chung, Y.-S. Jeong, D.-S. Park, Organizing a User-Created Computing Environment by RESTful Web Service Open APIs in Desktop Grids, *Journal of Internet Technology*, Vol. 15, No. 4, pp. 605-613, July, 2014.

[4] I. Gat, G. Succi, A Survey of the API Economy, *Business Agility & Software Engineering Excellence*, http://www.cutter.com/article/survey-api-economy-468936.

[5] APIs.io, http://apis.io/.

[6] mashape, https://www.mashape.com/.

[7] S.-P. Ma, C.-W. Lan, C.-H. Li, Contextual Service Discovery Using Term Expansion and Binding Coverage Analysis, *Future Generation Computer Systems*, Vol. 48, pp. 73-81, July, 2015.

[8] A. M. Zaremski, J. M. Wing, Signature Matching: a Tool for Using Software Libraries, *ACM Transactions on Software Engineering and Methodology*, Vol. 4, No. 2, pp. 146-170, April, 1995.

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

[10] M. Paolucci, T. Kawamura, T. R. Payne, K. Sycara, Semantic Matching of Web Services Capabilities, *The Semantic Web—ISWC 2002*, Sardinia, Italy, 2002, pp. 333-347.

[11] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, S. Hellmann, DBpedia- A Crystallization Point for the Web of Data, *Web Semantics: Science, Services and Agents on the World Wide Web*, Vol. 7, No. 3, pp. 154-165, September, 2009.

[12] G. A. Miller, WordNet: A Lexical Database for English, *Communications of the ACM*, Vol. 38, No. 11, pp. 39-41, November, 1995.

[13] C. Fellbaum, *WordNet: An Electronic Lexical Database*, MIT Press, 1998.

[14] R. Jonker, T. Volgenant, Improving the Hungarian Assignment Algorithm, *Operations Research Letters*, Vol. 5, No. 4, pp. 171-175, October, 1986.

[15] E. Stroulia, Y. Wang, Structural and Semantic Matching for Assessing Web-service Similarity, *International Journal of Cooperative Information Systems*, Vol. 14, No. 4, pp. 407-437, December, 2005.

[16] X. Dong, J. Madhavan, A. Halevy, Mining Structures for Semantics, *ACM SIGKDD Explorations Newsletter*, Vol. 6, No. 2, pp. 53-60, December, 2004.

[17] Y. Hao, Y. Zhang, J. Cao, Web Services Discovery and Rank: An Information Retrieval Approach, *Future Generation Computer Systems*, Vol. 26, No. 8, pp. 1053-1062, October, 2010.

[18] P. Plebani, B. Pernici, URBE: Web Service Retrieval Based on Similarity Evaluation, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 21, No. 11, pp. 1629-1642, November, 2009.

[19] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, 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, No. 1, pp. 17-39, January, 2005.

[20] D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, K. Sycara, Bringing Semantics to Web Services: The OWL-S Approach, *the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, San Diego, CA, 2004, pp. 26-42.

[21] D. Martin, M. Burstein, D. McDermott, S. McIlraith, M. Paolucci, K. Sycara, D. L. McGuinness, E. Sirin, N, Srinivasan, Bringing Semantics to Web Services with OWL-S, *World Wide Web*, Vol. 10, No. 3, pp. 243-277, September, 2007.

[22] J. Hu, X. Chen, Y. Cao, L. Zhu, A Comprehensive Web Service Selection Algorithm on Just-in-Time Scheduling, *Journal of Internet Technology*, Vol. 17, No. 3, pp. 495-502, May, 2016.

[23] J. Wu, Z. Wu, Similarity-based Web Service Matchmaking, *2005 IEEE International Conference on Services Computing (SCC'05) Vol-1*, Orlando, FL, 2005, pp. 287-294.

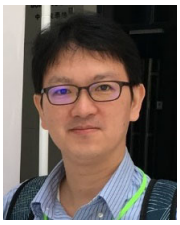
[24] M. Porter, *The Porter Stemming Algorithm*, http://www.tartarus.org/~martin/PorterStemmer/.

[25] Y. Li, Z. A. Bandar, D. McLean, An Approach for Measuring Semantic Similarity between Words Using Multiple Information Sources, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 15, No. 4, pp. 871-882, July- August, 2003.

Biographies

Shang-Pin Ma received his Ph.D. degree in Computer Science and Information Engineering from National Central University, Taiwan, in 2007. Dr. Ma is currently an associate professor in the Department of Computer Science and Engineering at National Taiwan Ocean University. His research interests include service-oriented computing, software engineering, mobile computing, and semantic web.

Hsuan-Ju Lin is currently a graduate student in in Computer Science and Information Engineering of National Taiwan Ocean University. Her research interests include service-oriented computing and software engineering.



Hsi-Min Chen is an Assistant Professor in the Department of Information Engineering and Computer Science at Feng Chia University, Taiwan. His research interests include software engineering, software architecture, service computing and distributed computing. Chen received his Ph.D. in computer science and information engineering from National Central University, Taiwan.

Ying-Jen Chen received her Bachelor (2013) and Master's (2015) degrees from the Department of Computer Science and Engineering, National Taiwan Ocean University, Taiwan. Her research interests include information retrieval, test-driven development, and service-oriented computing.

Wen-Tin Lee received his Ph.D. degree in Computer Science and Information Engineering from National Central University, Taiwan, in 2008. Lee is currently an associate professor in the Department of Software Engineering and Management at National Kaohsiung Normal University. His research interests include software engineering, service-oriented computing and software process management.