Camera Ready Paper PUB.CBS.EEB-EA-39907-1

Searching services "on the Web": A public Web services discovery approach

Chen Wu and Elizabeth Chang
Digital Ecosystems and Business Intelligence Institute
Curtin University of Technology, Perth 6845, Australia
{Chen.Wu, Elizabeth.Chang}@cbs.curtin.edu.au

Abstract

With the surge of Service-Oriented Architecture (SOA) and Web services, service discovery has become increasingly crucial. Public Web services that are available "on the Web" provide unlimited values for a great number of online service consumers and developers. However, the public UDDI Business Registry – the primary service discovery mechanism over the Internet – has been shut down permanently since 2006. This has raised serious problems for public Web services discovery. In this paper, we propose a WSDL-based public Web service discovery approach. It provides a simple-to-use yet effective Web service discovery mechanism that can retrieve relevant Web services directly from the Internet to suffice various requirements. The major contributions of this paper include: (1) a review on the state-of-the-art WSDL-based Web services discovery approaches; (2) the unique WSDL crawling and manipulation algorithms catering for the Vector Space Model, and (3) a proof-of-concept prototype – an online Web services search engine – based on the proposed approach.

1. Introduction

Gartner [1] has reported that, by the year 2010, 80% of application software profits growth will be driven by products based on Service-Oriented Architecture (SOA). This will bring all the SOA benefits such as loose-coupling, flexibility, process composability, etc. into most commercial software products. Moreover, with the rising evolution of SaaS (Software-as-a-Service), increasing software or even hardware will fit into the SOA paradigm. Various service users (e.g. designers, developers, consumers, etc.) need to know, in a cost-effective manner, what kind of services is available "on the Web", what their detailed capabilities are, and how they can be used under different business contexts. This has made service discovery of paramount importance. Service discovery paves the way for conducting further important SOA activities such as service binding, sharing, reusing, and composing in a dynamically changing business environment.

However, the public UDDI Business Registry – the primary service discovery mechanism over the Internet – has been shut down permanently since January 12, 2006[1] due to several reasons [2-5]. This has made the most important public Web services discovery mechanism missing from the Web services community. It does not mean public service registry will totally give way to "private registries". As suggested in [6], the central registry will continue to be a universally-known reference for many companies that "cannot serve a request locally". This is because most private registries would focus on a specific, closed domain such as the corporate network. Hence, in an open, loosely-coupled, demand-driven business environment, a dedicated public Web service registry is still essential to SOA.

In this paper, we will propose a WSDL-based Web service discovery approach, aiming to provide simple yet effective Web service discovery mechanism that can retrieve relevant Web services from the Internet to serve a wide range of users such as service consumers, service developers, service deployers, and service brokers. The solution leverages well-established research models from research fields including: Information Retrieval (IR), Natural Language Processing (NLP), XML Retrieval, etc. in order to fully utilise texts contained in the service description – WSDL files.

The rest of this paper is structured as follows. Section 2 presents the conceptual framework and details the service discovery approach. Evaluation results are presented Section 3. Section 4 provides a succinct review on the related work. The paper concludes in Section 5.

[1] http://uddi.microsoft.com/about/FAQshutdown.htm

2. The Discovery Approach

The overall service discovery approach is illustrated in Figure 1 consisting of four major steps. Due to the page limit, we will focus on the first two steps in which original work has been carried out. The last two steps can be realised using many existing Vector Space Model (VSM)-based search engine techniques. The basic idea is thus to convert the service discovery problem into the classic VSM indexing and searching problem while reserving unique features (e.g. structural information) in WSDL documents.

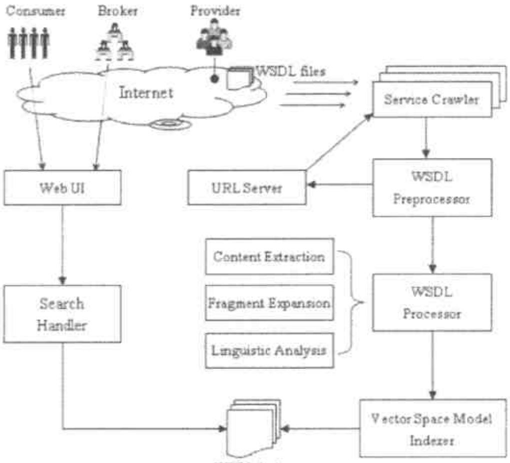


Figure 1 The overall approach

2.1. WSDL Crawling and Preprocessing

Crawling is an essential function for a Web search engine as it provides primary input for indexing and searching. In general, web crawlers utilise the graph structure of the Web to move from one page to another. When thinking of applying Web crawling techniques into WSDL, several characteristics distinguish the crawling algorithm. First, there are no links in WSDL files that can connect one WSDL file with another. Aiming at describing a single Web service, WSDL is not meant to be used for connecting different Web services. Hence, the crawler cannot simply generate more WSDL pages out of several seeds WSDL files. This problem has made crawling WSDL files become difficult and different from existing crawling approaches relying on 'graph search problem' analysis [7]. Moreover, assuming the link mechanism can be solved at the later stage, how to determine the initial seed pages that can later produce more WSDL files remains an open issue.

Previous studies [4, 5] have suggested that three discovery candidate sources can be utilised for service retrieval: UDDI Business Registry, public Web services registries, and customised commercial search engines. As indicated by existing studies and practices [2, 3, 5] and [4], UBR has several critical problems (e.g. unknown ranking mechanisms, missing verification and business models, the 'experimental' nature, etc.), which have made "UDDI registries are not good start" [5]. Moreover, the permanent shutdown of the public UDDI has made UBR unavailable to start with.

Public Web services registries are online service registries that usually do not comply with the UDDI specification. According to [5], these online registries in fact host more available Web services that can be remotely invoked and used by service consumers. Therefore, they are more widely used amongst Web services development communities. The main problem is that they do not include Web services which are available but not yet registered. Moreover, the numbers of registered services provided by these registries are "vague and imprecise" [4]. The quality of these registered Web services can be worse given that only 60% of these registered services are valid, leaving almost half of these registered Web services unreachable [5].

Customised commercial search engines generally provide large numbers of online Web services in the form of WSDL files. For example, Google indicates that over 25,600 WSDL files are found from its index when searching for "FILETYPE:WSDL". However, the validity of these WSDL files cannot be guaranteed as Google does not support WSDL file format. Hence, many non-WSDL (e.g. HTML) files are also included in the result lists simply because their file names end with ".WSDL". According to our experiment (Figure 2), only 12% of Google results are invalid WSDL files. This does not include many returned URLs that cannot reference to a file representation at all. Moreover, due to the restricted use of official Google API, only the

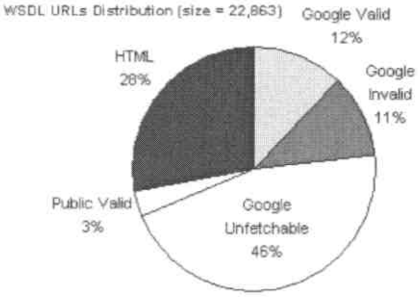


Figure 2 The conceptual model

first 1,000 files from the result list are available for the same query (e.g. "FILETYPE:WSDL"). Thus, technically, one can only obtain approximately 120 valid WSDL files from Google's huge (nearly 10 billion) web page index.

Therefore, we have employed a hybrid approach that utilises both public Web services registries and the Google search engine. First, we directly collect WSDL file sources [5] with roughly one thousand valid WSDL files. For Google, we designed an algorithm that utilised an English dictionary to enumerate the word list that contains 57, 046 English word entries. For each word entry, it conducts the Google query, attempting to search all relevant WSDL-like files that also contain this word as evaluated by Google. Theoretically, this will bring totally 57, 046 × 1,000 WSDL files from Google. However, in practice, most word entries do not return any results, and most words have returned less than 10 files. Moreover, many URLs are returned more than once under different word entries. The algorithm has returned totally 22,103 URLs (Figure 2) that can potentially generate WSDL files. Compared to 25,600 files reported in [4], this algorithm has covered approximately 86% of the entire WSDL files collection indexed by Google.

The next step is to dereference each URL in order to fetch the content of the WSDL file. In our algorithm, a number of independent crawlers work in parallel to download WSDL files based on URLs fed by the URLServer. All crawlers are available in the *CrawlerPool*, which reuses idled crawlers in order to control the resource consumption. Each crawler is realised as an autonomous working thread. The algorithm first retrieves a list of URL and allocates them to each crawler as a separate job. For each job, the crawler fetches the WSDL file given the URL and saves it to a specified location. In our experiment, this algorithm has fetched totally 10,255 WSDL files.

WSDL Preprocessing is crucial in that many WSDL files from Google are invalid. Therefore, we designed an algorithm to scrutinise these WSDL files. We then categorise these invalid WSDL files into three different groups. This way, we can afterwards conduct an analysis on the nature of these invalid WSDL files, which in turn can improve the crawling algorithm. At this stage, this is particularly useful for discovering more WSDL links out of HTML files that had been mistakenly identified as WSDL files by Google. Each WSDL file is parsed by a regular WSDL parser. When encountering invalid WSDL files, the algorithm checks whether it appears to be a HTML file. If yes, it is then sent to an HTML parser in attempt to find out some HTML links that might point to 'real' WSDL files. These potential WSDL links are sent to *URLServer*, waiting for the next round of crawling.

Based on results collected from both *crawling* and *preprocessing*, we provide the WSDL file distribution in Figure 2. The number of WSDL URLs is 22,863, of which Google focused crawling provides 22,103 URLs, the remaining 760 URLs come from [5]. The original data source in [5] was reported to contain 1,544 valid WSDL URLs. However, some of them duplicate with the URLs obtained from Google. Moreover, some are not reachable (e.g. all services from the registry 'Salcentral.com' are not available anymore) during the time of our experiment. Overall, this leaves only 760 valid public Web services registry-based URLs in our experiment, which amounts to 3% of the total URLs.

As shown in Figure 2, only 12% of the URLs point to retrievable and valid WSDL files. This number is almost the same as invalid WSDL URLs (11%). This means that nearly half of the 'real' (vs. HTML) WSDL files on the Internet are invalid, putting the quality of public WSDL files in to a question. Future research can be done in order to gain better understanding of the reasons for the half invalid WSDL files. As the largest hitherto Web search engine, Google has returned 6439 HTML files where WSDL files are desired, taking up to 28% of the whole sample URLs. This is because Google has not yet rigorously indexed the WSDL files, including all HTML files with ".WSDL" suffix in the result list. However, these HTML files provide entries to many 'hidden' WSDL files connected by HTML links. A very promising research direction is thus to analyse all these 'WSDL portal-like' HTML pages and to infer the relation between them and WSDL files in order to create the WSDL focused crawling without using Google. Finally, it is clear that almost half (46%) of the URLs are not reachable during the time of fetching. This problem can be caused by many factors such as slow network connection, DNS failure/timeout, firewall restriction, or even temporarily server shutdown, etc.

2.2. WSDL Processing

2.2.1. Content Extraction. A decision has to be made as to which XML nodes (e.g. elements, attributes, comments, etc.) will be extracted. Let us consider the stock price WSDL[2] in WSDL 1.1 specification as an example. Presumably, the name attribute of an XML element represents the semantics for that particular element. Hence, the value (i.e. "nmtoken") of attribute

[2] Due to the page limit, refer to http://www.w3.org/TR/wsdl#_style

"name" is important. For example, the value (e.g. "GetLastTradePrice") of the "name" attribute for element <operation /> carries useful information implying the purpose of this operation at the lexical level. Therefore, values (i.e. "nmtoken") of attributes "name" are extracted from a number of important WSDL elements – "definitions", "message", "part", "portType", "operation", "input", "output", "service", and "port". Moreover, the value of "uri" attribute for element "targetNamespace" is also extracted to identify the service provider. It can also be used for matching two "nmtoken" from different WSDL files but with the same namespace. For the implementation purpose, the WSDL file needs to be parsed into object model, which in turn gives rise to all the extracted value of name tokens.

Note that the value (i.e. "qname") of attribute "element" for element "part" is also extracted for capturing the data structure of the parameters sent to and from the service operations. This leads to the recursive extraction of underlying data types for this element and/or type. In this example, the value (i.e. "body") of attribute "name" for element "part" gives little useful information representing the real meaning of the input message part. Nevertheless, values (i.e. "TradePriceRequest" and "TradePrice") of attribute "element" provide very valuable data in understanding the meaning of two message parts. Delving into the data structure of these two elements defined within the WSDL element "types" and "schema", one can find more important lexical information of these two message parts by extracting the value of attribute "name" for element "element". Thus, in the case of "TradePriceRequest", the element value is "tickerSymbol". For "TradePrice", "price" is extracted. Thus, by solely exploring lexical information, one can speculate that this Web services takes as input the stock *ticker symbol*, and returns as output the *price* of the corresponding stock. The related operation name "GetLastTradePrice" also supports this proposition.

2.2.2. Fragment Expansion. In preparing XML documents for full-text information retrieval, Lehtonen [8] has raised two major issues – *fragment selection* and *fragment expansion*. Section 4.2.1 has mainly dealt with the issue of *fragment selection*, where "nmtokens" in various WSDL elements have been extracted for further processing. In this section, we focus on the second issue *fragment expansion*, a process during which plain texts extracted from XML nodes are manipulated (e.g. addition, removal, reordering, etc.) in order to improve the retrieval precision. The basic idea of fragment expansion is to retain some important structure-related information before XML markup is removed to produce plain texts. This is motivated by the concern that XML-stripped texts often have little metadata (e.g. structural) information that, if otherwise presented, could be crucial in determining the retrieval precision. Following this idea, we define as follows two primary objectives of *fragment expansion* (FE) for WSDL-based service retrieval.

First, FE shall provide different weights to term occurrences at different WSDL locations. For example, "nmtokens" appearing in some WSDL elements are generally more significant than others in expressing the overall capability of a Web service. Consider two Web services A and B. Suppose A can provide comprehensive weather information (e.g. temperature, humidity, wind, waves, trend, climate comparison, etc.) for geographic regions all over the world. Service B, on the other hand, focuses on delivering local tourism information such as hotels, flights, restaurants, banks, and weather forecasts. It is clear that Web service A is more relevant than B in response to service requests such as 'weather report'. However, this cannot be guaranteed since the frequency of the word 'weather' in Service A might be exactly the same as in Service B. Second, FE shall retain some important structural information to cater for service retrieval at various levels of granularity. It is evident that converting all token names from WSDL elements into plain texts will eliminate all structural information, which sometimes determines the service relevance to a user query. This is particular the case when searching the WSDL element "<operation />", where words in an operation name constitute a complete independent business meaning.

In order to achieve these two objectives, we have utilised three techniques introduced in [8] – *Reduplication*, *Reordering*, and *Addition*. *Reduplication* is useful to achieve the first objective of FE – i.e. to provide different term weights based on its WSDL locations. As analysed earlier, tokens in elements such as "<service />" and "<porttype />" are generally more important than others in describing the overall capability of a Web service. Therefore, these tokens are reduplicated in order to increase their relative weights by doubling the raw frequency of tokens in important elements. Since the whole tokens will be duplicated, duplication will not break the original term sequence in name tokens.

Reordering of WSDL elements and name tokens is a strategy to delimit the phrase found in the word sequence. This is helpful in separating "<operation/>" from other WSDL elements in order to retain self-descriptive phrases in "<operation/>" name tokens.

Elements in WSDL files are reordered so that all "<operation/>" elements are placed at the end of WSDL files. This way, tokens used in "<operation/>" are separated and distinguishable from other elements. On the other hand, *reordering* is also useful to bring structurally-related WSDL elements 'closer' in the plain texts for the purpose of proximity queries. For example, the "<part />" element "TradePriceRequest" needs to be relocated to the place right before the data type element "tickerSymbol". As a result, a phrase query such as 'trade symbol' is more likely to retrieve this service due to the proximity between these two words. In a nutshell, the appropriate *reordering* that rearranges the original positions of WSDL elements can improve service retrieval precision.

Addition provides more subtle impacts in separating phrases from the same type of WSDL elements. A Web service often has several operations. It is still problematic to differentiate terms from two adjacent operations once all XML markups have been removed. *Addition* aims to insert extra metadata such as position increment information before and after each occurrence of the operation name tokens. Consider an example in which a Web service has three operation tokens: "upload Image", "decode MP3", and "download Music". *Addition* will conceptually convert tokens into manipulated texts as: "{3} upload {1} image {3} decode {1} MP3 {3} download {1} Music {3}". Here, number "{1}" stands for the default one position increment between two successive terms during the tokenisation. The number "{3}" are artificially placed to represents three position increments, which imply a big gap between two terms.

2.2.3. Linguistic Analysis. Once extracted and expanded, these raw name tokens cannot be utilised directly due to various reasons such as the sub-language patterns, machine-generated code, or programming conventions, etc. Therefore, they need to be converted to natural languages before being indexed using IR models. The most important [3] linguistic technique is *tokenisation*, by which a name token is split into a sequence of smaller meaningful terms by a special tokeniser. For example, the operation name "GetLastTradePrice" shall be tokenised into four sequential terms – "Get", "Last", "Trade", and "Price". Seemingly, such a tokenisation appears as straightforward as to detect the lower case letters and upper case ones in the name token. However, the problem can become more complicated when considering name tokens such as "GetMyeBayServices", "AUD2USD", or "downloadMP3Music". For example, applying the simple capital letter rule for the first token gives the result "Get", "Mye", "Bay", and "Services". This is not desirable since the company name "eBay" is mistakenly split into two different terms. As a result, a service retrieval on "eBay" cannot match this operation. Take the second name token as another example. It is acceptable to generalise a rule that uses digit (e.g. "2") as the separator to delimit acronyms "AUD" and "USD". However, this rule does not work properly for the third name token, which will be unfortunately decomposed into separate terms – "download", "MP", "3", and "Music". Hence, it will miss out all user queries on "MP3", the feature all this operation is about.

The tokenisation is based on the *Maximum Matching Algorithm* (MMA), which has been widely used for Chinese segmentation studies [9, 10]. The basic idea of MMA is to use an external word list (e.g. a Chinese dictionary) to verify the possible word tokens parsed out from the unsegmented text. The algorithm starts from the first character in a text and reads in one character at a time to form the 'character sequence' ***cs***. After reading each character, it attempts to find in the word list the longest word ***w*** that starts with the current character sequence ***cs***. If ***w*** can be also found in the text (i.e. the remaining part of ***w*** also matches the following characters read from the text), the MMA marks a boundary at the end of ***w*** and starts again from the following character using the same longest word matching strategy until reach the end of the character sequence. If none matching words can be found in the word list, the first character in the character sequence ***cs*** itself will be identified as a single word. Table 1 demonstrates some tokenisation results.

Table 1 tokenisation results

WSDL Names	WSDL Tokens
Downloadi2Profile	download, i2, Profile
findeCommerceWebsite	find, eCommerce, Website
getDNACombo	get, DNA, Combo
editP3PPolicies	edit, P3P, Policies
getADSL2Specification	get, ADSL2, Specification
submitAdhocQuery	submit, Adhoc, Query
arg0	arg, 0

2.3. WSDL Indexing

Indexing refers to the process of creating and maintaining such a critical data structure, which allows fast searching over large amounts of data. It takes as

[3] due to the page limit, we omit other techniques here

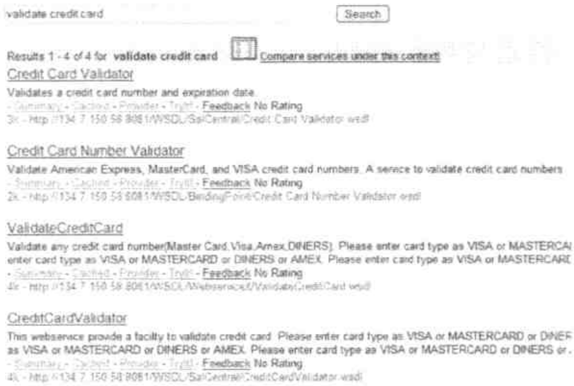
inputs tokenised and lemmatised terms with their associated occurrences information in each document and generates as outputs the compiled data arrangement with pre-aggregated information optimised for fast searching. The data structure of inverted index is consistent with the notion of *term-document* matrix, which consists of term vectors as matrix rows and document vectors as matrix columns.

2.4. WSDL Searching

During the service retrieval, the well-crafted index data structure will be looked up and fully utilised in order to find WSDL files containing words in the query from service consumers and brokers. In a typical service retrieval scenario, a service consumer submits his or her service query via the Web user interface to search intriguing Web services. In response, the service retrieval module (e.g. a search engine) analyses and converts the query into tokens, which are then compared with terms in the inverted index in order to find term occurrences. The list of occurrences is returned to the service consumer through the Web UI. [11] has generalised three major steps for searching on an inverted index – *Vocabulary search*, *Retrieval of occurrences*, and *Manipulation of occurrences*. Therefore, we have built the process of service retrieval based on these three steps. Furthermore, an extra effort has been conducted for service relevance ranking based on the *Cosine Similarity*. We have thus added this important VSM activity as the fourth step – *Ranking of occurrences*. The search engine UI and the sample searching result is shown in Figure 3.

3. Evaluation

In this section, we provide the evaluation result from our experiments regarding the matchmaking performance and the scalability of the search engine. To test the matchmaking performance, we carry out 10 queries that contain the most frequently entered query terms that had been captured by the search engine's logging system. We then measure the Precision and Recall respectively. We finally calculate the average precision values [11] at each recall level as shown in the Precision-Recall curves as shown in Figure 4. For the scalability test, we measure the average time duration and memory consumption during the searching process given the increasing number of service consumers. The result is shown in Figure 5. It can be seen that both time and space complexity are linear and thus the behaviour of the search engine is not sensitive to changes in system loads.



Results 1 - 4 of 4 for **validate credit card** Compare services under this context!

Credit Card Validator
Validates a credit card number and expiration date.
- Summary - Cached - Provider - TryIt - Feedback No Rating
3k - http://134.7.150.58:8081/WSDL/SaiCentral/Credit Card Validator.wsdl

Credit Card Number Validator
Validate American Express, MasterCard, and VISA credit card numbers. A service to validate credit card numbers
- Summary - Cached - Provider - TryIt - Feedback No Rating
2k - http://134.7.150.58:8081/WSDL/BindingPoint/Credit Card Number Validator.wsdl

ValidateCreditCard
Validate any credit card number(Master Card,Visa,Amex,DINERS). Please enter card type as VISA or MASTERCA
enter card type as VISA or MASTERCARD or DINERS or AMEX. Please enter card type as VISA or MASTERCARI
- Summary - Cached - Provider - TryIt - Feedback No Rating
4k - http://134.7.150.58:8081/WSDL/Webservicex/ValidateCreditCard.wsdl

CreditCardValidator
This webservice provide a facilty to validate credit card. Please enter card type as VISA or MASTERCARD or DINEF
as VISA or MASTERCARD or DINERS or AMEX. Please enter card type as VISA or MASTERCARD or DINERS or .
- Summary - Cached - Provider - TryIt - Feedback No Rating
4k - http://134.7.150.58:8081/WSDL/SaiCentral/CreditCardValidator.wsdl

Figure 3 Result for searching "validate credit card"

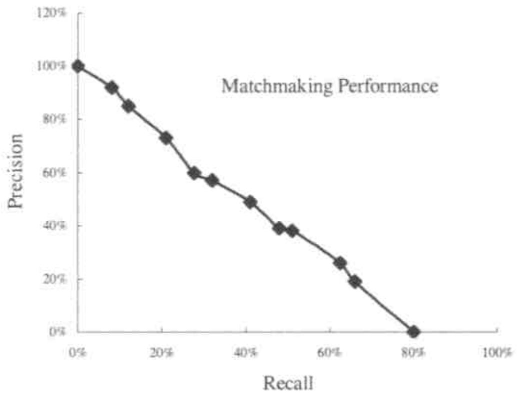


Figure 4 Result for searching "validate credit card"

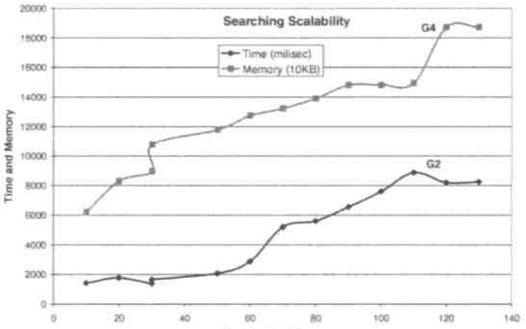


Figure 5 Searching scalability test result

4. Related work

In this section, we classify existing service discovery methods into two categories (Figure 6): WSDL-based and Ontology-based. The rationale is that

WSDL is the defacto standard for representing a Web service's functional capability and technical specifications "on the wire" [12]. It is then natural to discern service discovery methods that centre upon WSDL with those do not. It should be noted that these two categories are not absolutely orthogonal with each other. For example, in the ontology-based method WSDL-S [13], annotation have been made to reference to a domain ontology through the standard WSDL extension mechanism. Hence, we define that WSDL-based refers to those methods that take regular WSDL files 'as-is' without further augmenting. Ontology-based methods, on the other hand, aim to provide a 'semantically enriched' version of WSDL files in order to automate complicated tasks such as service composition. In this section, we succinctly survey WSDL-based approaches. Interested readers can refer to [14] for an thorough understanding of ontology-based approaches.

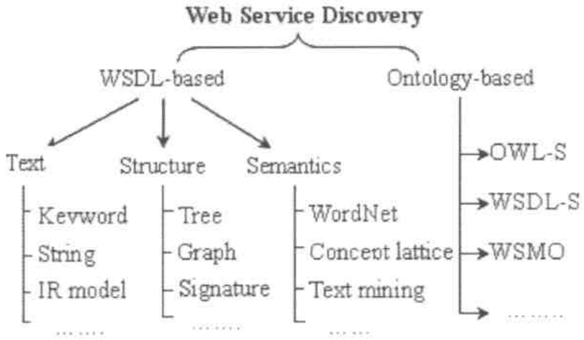


Figure 6 A summary of Web services discovery methods

Text-based method is the most straightforward way to conduct Web service discovery. The most widely used text-based is the keyword matching built in the UDDI public registry. The UDDI API allows developers to specify keywords of particular interests and it then returns a list of Web services whose service description contains those keywords. Beyond the literal keyword matching, research in XML schema matching ([15]) has applied various string comparison algorithms (e.g. *prefix, suffix, edit distance*) to match those interchangeable keywords but with slightly different spellings. This method is particularly useful for scientific Web services where many special terms, jargons, and acronyms are widely used in their service descriptions. For example, a bioinformatics Web service might have an operation called '*DNACombo*', which shall be relevant to a user search '*DNA Combination*'. The literal keyword method cannot tell the equivalence between *Combo* and *Combination*.

Similar to our work, several recent efforts have utilised IR models. Authors in [16] used the VSM to build a Web service search engine. [17] has attempted to leverage LSA, a variant of VSM, to facilitate web services discovery. However, both [16] and [17] rely on existing UDDI public registries. Hence, our work is different in that we have implemented a focused Web service crawling mechanism which does not exclusively rely on UDDI registries. Therefore, our experiment data set is purely obtained from the 'Web' with the public Web services nature. More importantly, different from [16] and [17], the texts used in our approach is extracted, analysed, and expanded directly from WSDL elements rather than service description written in natural languages. Unlike natural languages, WSDL is far more structural and compact. Towards that end, the VSM-method in [18, 19] has used the pattern of letter cases to split a long WSDL element into separate tokens. However, we have found such a heuristics is insufficient when facing a large amount of irregular, non-word WSDL terms and acronyms. Therefore, in our approach, we add a *WSDL Processor* component dedicated to deal with language and structural features of WSDL files.

WSDL with its embedded XML schema data types contains important structural information. Previous studies have attempted to use it to assist service discovery. For example, in [20], a WSDL file is treated as a structural tree that can be compared based on the structures of the operations' input/output messages, which in turn, is based on the comparison of the data types delivered contained in these messages. Likewise, the interface similarity defined in [21] is computed by identifying the pair-wise correspondence of their operations that maximizes the sum total of the matching scores of the individual pairs. More recently, the author in [19] also calculated the similarity of complex WSDL concepts given similarity scores for their sub-elements. Using the maximum-weighted bipartite matching [22] algorithm from the graph theory, the author defined a number of coefficiencies to determine the ultimate structural similarity score between two parts in a matching pair. Most of these WSDL structural matching methods are inspired from the signature matching [23], a software component retrieval method from software engineering research.

Although a standard WSDL does not provide semantic information, identifiers of messages and operations do contain information that can potentially be used to infer the semantics. This can be supported by an external lexical database such as the WordNet[4]. For example, when comparing two operations in [20],

[4] http://wordnet.princeton.edu

WordNet is used for deriving the synonyms for the semantic similarity calculation. The lexical similarity defined in [21] and [24] is also based on the concept distance computed from the WordNet sense hierarchy. Interestingly, research in [19] indicates that using WordNet may bring many false correlations due to its excessive generality. In this work, the author reports that VSM has achieved the overall best performance, outweighing the WordNet based semantic similarity method. The author also discussed that this might be caused by the ambiguity of the terms used in service specifications.

5. Conclusion

Service discovery is a key aspect in the SOA research community. In this paper, we have proposed a Web services discovery approach based on the public WSDL corpus and IR models. We first provided a focused literature review on the state-of-the-art WSDL-based Web services discovery approaches. We then present the conceptual model of our approach, which includes four essential steps: WSDL crawling, WSDL processing, WSDL indexing, and WSDL searching. While the first two steps produce the WSDL corpus using Web crawling and XML retrieval techniques, the last two steps leverage the Vector Space Model to conduct service indexing and searching. The approach also leads to a proof-of-concept prototype – the public Web services search engine. The matchmaking and scalability experiment results are also presented and evaluated. For future work, we are looking at introducing semantic mechanism (i.e. WordNet or Ontology) into the search engine in order to enhance the *recall* of the matchmaking performance.

7. References

[1] D. W. Cearley, J. Fenn, and D. C. Plummer, "Gartner's Positions on the Five Hottest IT Topics and Trends in 2005," *Gartner report*, vol. G00125868, 2005.

[2] D. Chappell, "Who Cares About UDDI?," Addison Wesley, 2002.

[3] U. Ogbuji, "UDDI 3.0? Who really cares?," Oreilly, 2005.

[4] D. Bachlechner, K. Siorpaes, D. Fensel, and I. Toma, "Web Service Discovery - A Reality Check," DERI, Galway 1/17/2006 2006.

[5] J. Fan and S. Kambhampati, "A Snapshot of Public Web Services," *ACM SIGMOD Record*, vol. 34, pp. 24 - 32, 2005.

[6] L. Baresi and M. Miraz, "A Distributed Approach for the Federation of Heterogeneous Registries," presented at Fourth International Conference on Service Oriented Computing, Chicago, USA, 2006.

[7] G. Pant, P. Srinivasan, and F. Menczer, "Crawling the Web," 2003.

[8] Lehtonen, "Preparing Heterogeneous XML for Full-Text Search," *ACM Transactions on Information Systems*, vol. 24, 2006.

[9] J.-S. Chang and K.-Y. Su, "An Unsupervised Iterative Method for Chinese New Lexicon Extraction," *International Journal of Computational Linguistics, Chinese Language Processing*, 1997.

[10] L. Tao, "Elektronische Tokenisierung fuer das Chinesische. Master thesis," Uni-muenchen., 2001.

[11] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*: Addison Wesley, 1999.

[12] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web Services Description Language (WSDL) 1.1," 2001.

[13] "Web Service Semantics," http://www.w3.org/Submission/WSDL-S/.

[14] J. Cardoso and A. Sheth, "Semantic Web Services, Processes and Applications," in *Semantic Web and Beyond: Computing for Human Experience*, R. Jain and A. Sheth, Eds.: Springer, 2006.

[15] H. H. Do and E. Rahm, "COMA - A system for flexible combination of schema matching approaches," presented at 28th VLDB Conference, Hong Kong, China, 2002.

[16] C. Platzer and S. Dustdar, "A vector space search engine for Web services," presented at Third IEEE European Conference on Web Services, Sweden, 2005.

[17] A. Sajjanhar, J. Hou, and Y. Zhang, "Algorithm for Web Services Matching," presented at APWeb, 2004.

[18] N. Kokash, W.-J. v. d. Heuvel, and D. A. Vincenzo, "Leveraging Web Services Discovery with Customizable Hybrid Matching," *Technical Report, University of Trento*, vol. DIT-06-042, 2006.

[19] N. Kokash, "A Comparison of Web Service Interface Similarity Measures," University of Trento 2006.

[20] E. Stroulia and Y. Wang, "Structural and Semantic Matching for Accessing Web Service Similarity," *International Journal of Cooperation Information Systems*, vol. 14, pp. 407 - 437, 2005.

[21] J. Wu and Z. Wu, "Similarity-based Web Service Matchmaking," presented at IEEE International Conference on Service Computing, 2005.

[22] L. Lovasz and M. D. Plummer, *Matching Theory*. North-Holland: Elsevier Science Publisher, 1986.

[23] A. Zaremski and J. Wing, "Signature Matching of Software Components," *ACM Transactions on Software Engineering and Methodology*, pp. 333-369, 1997.

[24] Z. Zhuang, P. Mitra, and A. Jaiswal, "Corpus-based web services matchmaking," presented at Workshop on Exploring Planning and Scheduling for Web services, Grid, and Autonomic Computing, 2005.