View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by espace@Curtin

©2008 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

An empirical approach for semantic Web services discovery

Chen Wu*, Elizabeth Chang*, Ashley Aitken*+
*Digital Ecosystems and Business Intelligence Institute
and +School of Information Systems
Curtin University of Technology, Perth 6845, Australia
{Chen.Wu, Ashley.Aitken, Elizabeth.Chang}@cbs.curtin.edu.au

Abstract

Component retrieval/discovery is a well-established research direction in Software Engineering. With the surge of Service-Oriented Architecture (SOA), service discovery has become increasingly crucial. However, the public UDDI Business Registry – the primary service discovery mechanism over the Internet – has been shut down permanently since 2006. Moreover, keyword-based service discovery is insufficient in coping with complex discovery requirements posed by modern software developers.

In this paper, we propose an empirical semantic-based Web service discovery approach. It provides an automatic Web service discovery mechanism that can locate relevant Web services based on concepts rather than keywords. The major contribution of this paper is three fold. First we articulate three requirements that software developers often raise during the component/service development and discovery process. Next, we propose the application of Latent Semantic Analysis into the area of Web services discovery. To our best knowledge, little work has been done in this area which leverages concept-based Information Retrieval models in service discovery. Last, we provide a proof-of-concept system prototype that can suffice three specific requirements of semantic service discovery.

1. Introduction

Semantics is the study of relations between the system of symbols (e.g. words, phrases, and sentences) and their meanings. Semantics play an important role in the complete lifecycle of Web services as it is able to help service development, improve service reuse and discovery, significantly facilitate composition of Web services and enable integration of legacy applications as part of automatic business process integration. Unfortunately, current Web Service Description Language (WSDL) standard operates at the syntactic level and lacks the semantic expressivity needed to represent the requirements and capabilities of Web Services. Moreover, industry standard UDDI only supports keyword and taxonomy-based service discovery, thus leaving out the semantics of Web services either.

This has motivated a great deal of research efforts towards the Semantic Web Services (SWS). Interested readers can refer to [1] for a comprehensive understanding of the SWS. The fundamental idea underlying current SWS community is that in order to achieve machine-to-machine integration, a markup language (e.g. annotation) must be descriptive enough that a computer can automatically determine its meaning. Following this principle, many semantic annotation markup languages for Web services have come into existence and use such as OWL-S [2], (formerly known as DAML-S [3]), and WSDL-S [4] that have gained great momentum in recent years. The main goal of both OWL-S and WSDL-S is to establish a framework within which service descriptions are made and shared.

The assumption of such an ontology-based markup language approach is that every SWS user (professional or end customer) is able to use a standard ontology, consisting of a set of basic classes and properties, for declaring and describing services. One concern [5] about this descriptive annotation-driven approach is its feasibility: since it would be much more time-consuming to create and publish ontology-annotated (WSDL) content as they would need to be done by domain human experts and powerful editing tools for common users. Other problems [6] might occur when different groups of users and communities want to manage the shared ontology. With this being the case, it would be much less likely for industry companies to adopt these practices as it would only slow down their progress.

1530-0803/08 $25.00 © 2008 IEEE
DOI 10.1109/ASWEC.2008.52

IEEE computer society

Authorized licensed use limited to: CURTIN UNIVERSITY OF TECHNOLOGY. Downloaded on March 31, 2009 at 00:22 from IEEE Xplore. Restrictions apply.

In this paper, we propose a semantic-based service retrieval solution using an empirical approach without relying on ontology engineering. This is achieved using the latent Semantic Analysis (LSA) method. Moreover, we are currently seeking effective ways that can convert some result in this paper – i.e. the higher-order term associations and the WSDL clusters semantic space – into lightweight ontology using some semi-automatic ontology learning methods. In other words, the approach proposed in this paper aims to pave the way upon which ontology-based annotation can be accomplished with less human involvement and at a faster pace. First, we would like to introduce some real development scenario that motivates the semantic-based service discovery.

Suppose a software developer is looking for a Web service offering basic arithmetic calculation for his taxation application. To locate a suitable service, the developer needs to provide with the search engine the criteria against which the service retrieval can compare with each indexed Web service. From a service consumer's perspective, the developer might try looking for services with the keyword 'calculator', which is perhaps the most straightforward term that come up in his mind. On the other hand, consider a service provider who has developed such a Web service that can indeed perform arithmetic calculation. To the understanding of this service provider, this service is somehow attached with the name "MathService" in its WSDL description, which also encloses a number of useful WSDL operations that the developer is actually after, such as "add", "multiply", "getSquareRoot", etc. Unfortunately, keyword-based service discovery method, the software developer is unable to find this service provided by the service provider. Such a limitation caused by the keyword-based matching requires service retrieval to measure the meaning or concepts inferred by a Web service rather than literal texts embedded in its WSDL document.

Alternatively, the developer might keep trying other related keywords when failing to find relevant Web services under the keyword 'calculator'. Even if he is able to locate some Web services that he thinks are relevant, he might still want to try some other terms which might potentially bring more appropriate Web services that suit his needs. This can repeat for several rounds until he feels assured that what he has found so far is the "best" result. It is clear that such a repetitive process is sometimes tedious and can be frustrating for many service consumers as the search engine is working in a passive 'request/response' mode. In other words, service consumers and brokers require the service retrieval to be smart enough to proactively provide some feedback or suggestion that can help the service consumer to find desirable Web services effectively and efficiently.

Finally, suppose the service retrieval has been equipped with the "concept-based" searching mechanism that seeks for the meaning instead of the keywords. It is quite likely that numerous (e.g. 100) Web services would appear in the result list when the developer try 'maths' since any services related to mathematics will be found out. It would be extremely time-consuming for the developer to go through each one of them and to choose the most appropriate one. Moreover, the developer might want different types Web services to solve a variety of maths problems. In this case, a semantic-based service clustering would be highly desirable for the developer to rapidly locate relevant Web services for more refined service retrieval requests. Indeed, such a service clustering mechanism is one of the key added values that can attract more service consumers to participate in the domain syndication.

The rest of this paper is structured as follows. Section 2 provides a review on the existing work. Section 3 presents the architectural design of our approach. Section 4 details the service discovery approach. Evaluation results are presented Section 5. The paper concludes in Section 6.

2. Related Work

Web services evolve from the concept of software component [7, 8]. An influential direction in component retrieval is the signature matching, where component are discovered based on their interface signatures. More specifically, signature matching exploits the structure conveyed in the interface definition of component as built-in information (i.e. type) in order to facilitate component retrieval. An advantage of signature-based component retrieval method is that it does not rely on additional knowledge (i.e. annotations, specifications, etc.) but the properties of the component only. WSDL is an XML format for describing Web services in terms of both logic abstraction operation and concrete network bindings [9]. Therefore, WSDL can be seen as an XML version of the interface definition language for Web services. This way, WSDL contains the important 'signature' information for Web services and thus can be used for signature matching.

In the seminal work of signature matching, [10] defined signature matching as "*the process of determining which library components match a query signature*". The signature of a component refers to the "*type of a function or the interface of a module*". The type here includes the list of types for the component's

Authorized licensed use limited to: CURTIN UNIVERSITY OF TECHNOLOGY. Downloaded on March 31, 2009 at 00:22 from IEEE Xplore. Restrictions apply.

input/output parameters and possible exceptions. For functional signature matching, [10] further defined two types of signature matching – the *exact matching* and the *relaxed matching* – in order to locate software functions from a software library.

An simple application of signature matching in Web services discovery can be found in [11], where the authors approach the automated process of Web services searching using the signature matching, i.e. the *Exact Match* and *Transformation Match*. Moreover, the authors also discussed a new signature match criteria – the *Contains Match*, in which the returned signature of WSDLs contain the types found in the search string. The indexing and searching in this work is based on the full-text searching mechanism, in which all the type information in the WSDL signature are pre-compiled (indexed) without considering their internal structures.

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 ([12]) 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 for Web services discovery. Authors in [6] used the Vector Space Model (VSM) to build a Web service search engine. [13] has attempted to leverage LSA, a variant of VSM, to facilitate web services discovery. However, both [6] and [13] 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 [6] and [13], 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 [14, 15] 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.

The use of LSA for Web services can be traced back to earlier component retrieval research. For example, authors in [16] has built a Java reuse repository using LSA for component retrieval. Similarly, research in [17] proposed an active component repository systems that support "reuse-within-development" using real-time LSA component retrieval.

3. The Overall Architecture

The architectural is illustrated in Figure 1. Initially, service providers deploy WSDL files via the Internet. Once deployed, service descriptions can be collected by a number of Service Crawlers, which constantly fetch WSDL files from the Internet. Crawlers hand over retrieved WSDL files and associated HTML files to the WSDL Preprocessor for further link analysis. This yields a list of new URLs that may point to some new WSDL files. These URLs are sent to the URL Server, which in turn initialises/reconfigures a new/idle Crawler for fetching the WSDL file referenced by each newly identified URL. All retrieved raw WSDL files are then passed to the WSDL Processor, which (1) parses WSDL files and extracts important data (e.g. operations, messages, data types, etc.), (2) analyses these data using certain linguistic methods such as tokenisation, lemmatisation, stopwords elimination, etc. The *WSDL Processor* generates the classical IR 'term document' which contains separated words in a flat structure. The term document is transferred to the VSM Indexer, which takes as the input all the term documents and generates as the output WSDL indices representing the term-document matrices. The indexing algorithm and VSM indices storage format has been discussed in our previous work. Interested readers can refer to [18] for an understanding of VSM-based service discovery.

In this paper, we focus on the LSA indexer, which takes as input the VSM indices and generates as output the semantic space for service retrieval. It is the most crucial component for semantic-based service retrieval.

Authorized licensed use limited to: CURTIN UNIVERSITY OF TECHNOLOGY. Downloaded on March 31, 2009 at 00:22 from IEEE Xplore. Restrictions apply.

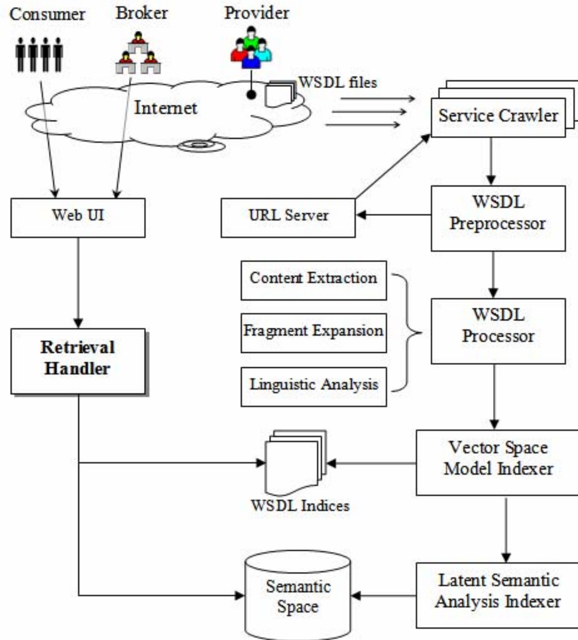

Figure 1 The architectural design

4. The Semantic Discovery Approach

4.1. Build WSDL Corpus

Based on results collected from both *crawling* and *preprocessing* [18], we have thus built the WSDL corpus that can yield the preliminary Term-WSDL matrix. According to the Zipf Law [19], a tiny amount of the words in a language are distributionally random and varied; the vast majority of words only occur in a very limited set of contexts. If one considers this WSDL corpus as a special language, and each WSDL file represents a context, one can reasonably conjecture that the Zipf Law applies in the WSDL corpus. Thus, it is expected that only a small number of terms appear in many contexts – WSDL documents; most terms only occur once or twice within the whole WSDL corpus. In order to verify this hypothesis, I have conducted a statistical experiment on non-zero elements in the conceptual term-WSDL original matrix.

In order to quantitatively verify the Zipf Law, we have collected the frequency and the rank for all the terms in the WSDL corpus and the results are shown in Table 1. For each term, we calculate its raw frequency in the whole WSDL corpus, i.e. how many times this term has appeared in the corpus. Based on the value of frequencies, we then sort terms in the descending order. This provides the ranking for each term (the first column). Note that in order to demonstrate the true distribution of the corpus language features, Table 1 has included stop words (e.g. http, soap, etc.) that will be eliminated during the linguistic analysis.

Table 1 Statistics for the term frequency and rank

Rank	Frequency	Terms
1	36033	get
2	11827	parameter
3	9540	soap
4	9347	http
5	6699	return
6	5915	body
7	5040	post
…	…	…
2093	12	academic
2094	12	sitename
…	…	…
6052	1	icalc
6053	1	depression
…	…	…

Using these two columns in Table 1, we generate the Zipf ranked distribution, where the X-axis represents the rank and the Y-axis depicts the frequency as shown in Figure 2. The distribution indicates mild concavity and a ranked exponent of 1: Zipf law, which can be roughly formulated as $freq(r) \sim r^{-1}$. Moreover, it is clear that a small number of terms such as "get", "parameter" are extremely 'popular' in the WSDL corpus. Table 1 indicates that less than 1% of terms have taken up more than 20% of all frequencies. Hence, based on our experiments, one can reasonably argue that the WSDL corpus crawled from the Internet follow the same pattern as normal discourse and natural languages in terms of the word distribution.

4.2. Construct WSDL Matrix

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 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. When constructing such a matrix, the first question is whether to normalise the WSDL vectors before creating the matrix. In order to preserve cosine similarities in the original space, one can length-

Authorized licensed use limited to: CURTIN UNIVERSITY OF TECHNOLOGY. Downloaded on March 31, 2009 at 00:22 from IEEE Xplore. Restrictions apply.

normalise the documents before conducting the Singular Value Decomposition (SVD). However, some research has shown that the additional use of the length of LSA vectors to be useful. This is because the length reflects how much was said about a concept rather than how central the discourse was to the concept. Therefore, we have made it a parameter in our experiment prototype– WSDL normalisation, i.e. whether or not to take into account the length of the WSDL vector.

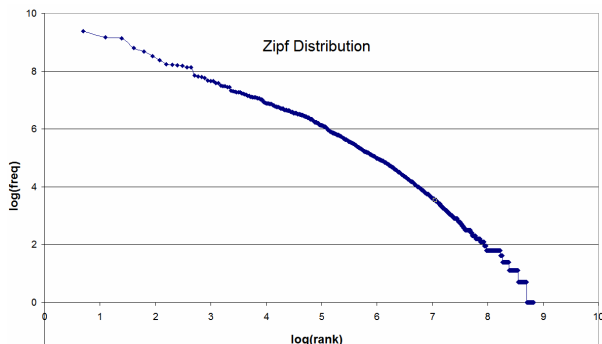

Figure 2 Zipf Distribution in the WSDL Corpus

In analysing the original matrix, we also provide a visualisation of the non-zero elements in the original sparse matrix as shown in Figure 3, where each nonzero element is marked as a coloured dot. Darker colour represents the larger value (i.e. the term weight) in the matrix ***A***'s entry. The figure shows that only 41,687 (less than 0.32%) entries are filled with nonzero values. This means most terms occur in a very few WSDL documents. In particular, several horizontal white "bands" shown in Figure 3 characterise the nature of the spare matrix ***A***. Moreover, it can be seen from Figure 3 that the number of vertical "coloured lines" are much more than the number of horizontal "coloured lines". Indeed, vertical lines form several clusters of coloured "bars". This observation resonances the fact that only a few words occur in many WSDL documents (i.e. the horizontal lines), and most terms appear only in limited set of WSDL files (i.e. the vertical bars).

The result shown here also coincides with the observation reported in [20], where the authors have found only a few WSDL parameters (e.g. "license key", "password") have been heavily used in many WSDL files, most parameters appear just once. In their work, only WSDL parameters (i.e. name attributes in "<types />" and "<part />") are considered as terms and their corpus contains only data source from [21]. However, the Zipf law still applies even in the corpus (language) with smaller words (all terms vs. parameters only) and contexts (3577 vs. 670). As a result, the original matrix ***A*** is a sparse matrix where most entries have zero as their values.

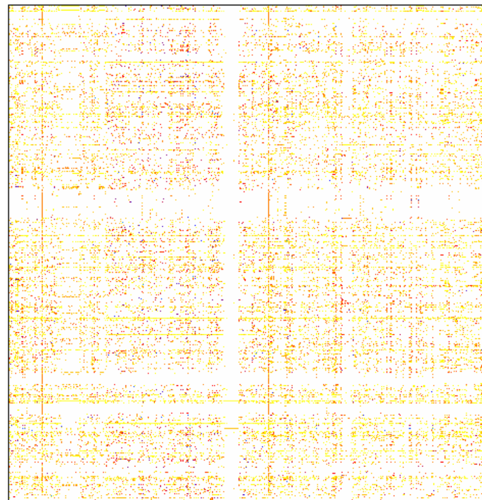

Figure 3 Sparse matrix A (row: terms 3,671, column: WSDL 3,577, nonzero 41,687)

When dealing with a matrix with thousands rows and columns, the memory consumption can be huge if the naive two dimensional arrays are used to represent the matrix in the memory. Moreover, the Zipf distribution indicates the matrix is very sparse as shown in Figure 3. Therefore, we have employed the Harwell-Boeing (HB) matrix [22] as a compressed representation of the original matrix ***A***. The HB sparse matrix file format is used to store a sparse matrix in a file. The space required to represent the matrix is reduced by using a compressed column storage format. If the matrix is read from the file into memory, it is common to use the same compressed column storage to represent the matrix in memory. This way, the total memory consumption for SVD can be dramatically reduced. For example, in our experiment, it costs merely 900K RAM to store a 3671×3570 HB matrix.

4.3. Conduct Singular Value Decomposition

Traditional SVD algorithms that apply orthogonal transformations directly to the sparse matrix ***A*** often requires tremendous memory consumption, and hence is not scalable when the terms and WSDL documents become larger. This statement is supported during our initial SVD experiment, where the JAMA (a Java Matrix Package[1]) is used to process the SVD. The JAMA library takes unacceptable long time (14 hours), trying to solve the SVD for a 3671 (terms) × 3577 (WSDL) matrix before it eventually throws the "out of

[1] http://math.nist.gov/javanumerics/jama/

Authorized licensed use limited to: CURTIN UNIVERSITY OF TECHNOLOGY. Downloaded on March 31, 2009 at 00:22 from IEEE Xplore. Restrictions apply.

memory" exception given the maximum Java heap has been manually set to one gigabytes.

In order to tackle this issue, we have used the large scale sparse SVD method proposed in [23] as the SVD algorithm in this paper for semantic service retrieval. The basic idea is to convert the SVD problem into an eigenvalues problem for a symmetric matrix, which has been well studied and can be solved using numerous canonical sparse symmetric eigenvalue solutions. Formally, given a $m \times n$ matrix $\boldsymbol{A}$, we aim to construct a symmetric matrix $\boldsymbol{B}$ associated with $\boldsymbol{A}$, such that the SVD of $\boldsymbol{A}$ can be obtained from the eigenvalues and eigenvectors of the matrix $\boldsymbol{B}$. Berry [23] has given two methods to construct such a matrix $\boldsymbol{B}$. In the first method, a $(m+n)\times(m+n)$ matrix $\boldsymbol{B}$ is constructed as Equation 1. Cullum and Willoughby [24] have proved that the SVD of $\boldsymbol{A}$ can be obtained from the eigenvalues and eigenvectors of the matrix $\boldsymbol{B}$ in Equation 1. The second method constructs a $n \times n$ matrix $\boldsymbol{B}$ as shown in Equation 2. Berry [23] has also demonstrated the fundamental relations between eigenvalues of $\boldsymbol{B}$ (Equation 2) and the SVD of $\boldsymbol{A}$. That is, the singular values in S are the nonnegative square roots of the n eigenvalues of $A^T A$, and the first r columns of S and T are orthonormalised eigenvectors corresponding to the r nonzero eigenvalues of AA^T and $A^T A$ respectively.

$$B = \begin{pmatrix} O & A \\ A^T & O \end{pmatrix}$$

Equation 1

$$B = A^T A$$

Equation 2

In this paper, we have chosen the second method to conduct the SVD as it is easy to prove and understand from the matrix theory that the singular values of the real symmetric matrix B are the absolute values of its nonzero eigenvalues (see [25] for a simple proof). The eigenvalue problem is implemented using a variant (i.e. "las2" [26]) of the single-vector Lanczos algorithm [27] in order to adapt to the matrix $\boldsymbol{B}$ defined in Equation 2. The basic idea of Lanczos algorithm is to generate a series of tradiagonal matrices T_j, such that the extremal eigenvalues of each $j \times j$, T_j are progressively better estimates of the eigenvalues of $B = A^T A$. Once these sequence is generated, select some T_k and compute its eigenvalues, which are the approximation of the eigenvalues of $\boldsymbol{B}$, and hence the singular values of $\boldsymbol{A}$. The corresponding singular vectors can be approximated by obtaining the corresponding eigenvectors of these eigenvalues that satisfy $T_k v = \lambda v$. Detailed mathematics supporting this algorithm can be found in [23].

4.4. Generate Index

Once the SVD and its truncation are achieved, the result needs to be persisted on to the storage so that it later on can be used by the retrieval process. This process is defined as generating the SVD index. The SVD index has to cater for the output of SVD truncation result, which is written to the index. Therefore, all data within the SVD result need to be efficiently saved onto the storage. Next, the index is served for service retrieval, thus the data shall be easily read and captured by the retrieval process for various purposes (e.g. similarity comparison). Last, the SVD index data structure needs to be compatible with existing VSM. This way, meta-data from VSM can be easily referenced to and any changes made in VSM can be timely updated in SVD index as well.

The output of SVD is can be re-written as the dyadic decomposition form:

$$A_k = \sum_{i=1}^{k} t_i s_i d_i^T ,$$

Equation 3

where t_i and d_i are column vectors of T and D respectively, and $1 \le k < rank(A) = r$. Therefore, the output contains k sets of triplets $\{t_i, s_i, d_i\}$, which are to be kept onto the storage. This provides basic requirements for "writing" part of the logic data structure of the SVD index. In the meantime, row vectors of T and D are also very important as they determine the similarity between terms, documents (WSDL), terms and documents. This can be considered as the "reading" part of the SVD index. To stay compatible with VSM, the SVD index "extends" the data structure of VSM through referencing to two entities.

Figure 4 illustrates the logic data structure of the SVD index. The design places the "reading" as the first priority since the ultimate goal of SVD indexing is to serve the service retrieval. The time complexity of service retrieval is thus more important than the one for writing SVD result. In other words, the data structure is optimised for efficient reading while guaranteeing effective writing. The data structure includes two parts – the VSM proxy (top) and the SVD triple (bottom). The VSM proxy is the interface between the VSM and the SVD in order to fulfil the requirement of interoperating between VSM and SVD. It contains a small set of VSM data attributes (e.g.

Authorized licensed use limited to: CURTIN UNIVERSITY OF TECHNOLOGY. Downloaded on March 31, 2009 at 00:22 from IEEE Xplore. Restrictions apply.

Term Value) copied from the original VSM data structure and is referenced by SVD triples for meta-data access. The VSM proxy contains two entities, representing the VSM Term Vector and The VSM WSDL Vector respectively. Processes that deal with SVD might not be aware of the existence of VSM indices as VSM proxy is the only external sources they will start from and reference to.

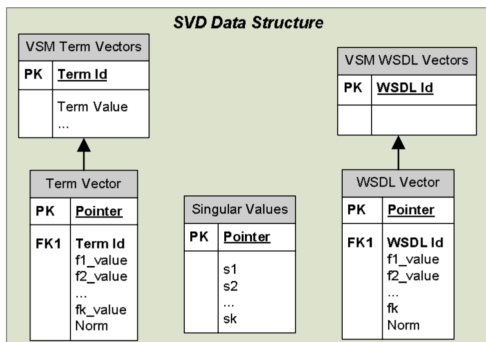

Figure 4 Logic data structure of the SVD index

The SVD triple contains three entities based on Equation 3, where t_i and d_i are conceptually column vectors. This requires the physical storage of their vector components to be consistent with such a column-based order. In a relational database, this can be implemented as filling row values under a particular column, e.g. the "f1_values" for all terms. In the file system, this mechanism can be implemented as an address pointer that jumps every k storage units to allocate each value under each column. Such a seemingly inefficient storage strategy however increases the efficiency for service retrieval, where it is the row vector that needs to be frequently compared and is of great concern by most service retrieval requirements. Therefore, all the row vectors components are stored in a continuous manner in order to suffice the "reading" requirement at the cost of compromising the "writing" requirement. The singular values are stored in a row for the quick access and reference. It should be noted that, the logical data structure illustrated in Figure 4 can be realised on various physical storage such as file systems, RDBMS or even the in-memory RAM.

4.5. Service Searching

Service searching is the basic mechanism by which service consumers and brokers can find their desirable Web services. Similar to VSM-based service searching, service consumers submit their query via the Web user interface and expect a ranked list of Web services based on their relevance to the query. Unlike VSM, in LSA-based service searching, this ranked list is no longer a "list of occurrences". It is the concept (i.e. semantics) rather than the literal keyword that determines their relevance and hence their ranking. This is because the term-by-document matrix A_k from SVD has captured the higher-order association between terms and WSDL documents. Each WSDL document is projected onto a rank-reduced semantic space, where only 101 (vs. 3671) dimensions are used to 'feature' the characteristics of the WSDL document. In this section, we will discuss the detailed searching process based on the SVD indexing result – the matrix A_k.

The first question is to decide whether the WSDL vector shall be normalised again. The normalisation here is after SVD. This boils down to whether the cosine or the inner product shall be used as the score function. The score function computed in the reduced dimensional space is normally the cosine between vectors. Empirically, this measure tends to work well, and there are some weak theoretical grounds for favouring it [28, 29]. Therefore, we have decided to use the cosine as the default similarity function to measure the similarity between a query vector and a WSDL vector. The second question is that whether the query vector needs to be scaled by the singular values before calculating the cosine similarity. The searching process is illustrated in the pseudo code (see Table 2).

Table 2 Service Searching Pseudo Code

```
10.   Input query : String
20.   Output a ranked list of Web services based on
      their relevance to the query
30.
40.   qv : QueryVector = initialiseQV(k) //all
      components are set to zero, k-dimensional
50.   sv: SingularValues = readSingularValues(k)
60.   FOR EACH term in query
70.     termVector := readTermVector(term)
80.     w := calculateWeight(term)
90.     FOR EACH t in QueryVector
100.      i := the index of t in termVector
110.      t := t + (w * termVector[i] * 1 / sv[i])
120.  queryNorm := qv.calculateNorm()
130.
140.  rs : = new SearchResultSet(THRESHOLD)
150.  FOR EACH wsdl in WSDLCorpus
160.    wv := readWSDLVector(wsdlId)
170.    wsdlNorm := wv.readNorm()
180.    FOR (int k = 0; k < nfactor; k++)
190.      sum := sum + wv[k] * qv[k]
```


Authorized licensed use limited to: CURTIN UNIVERSITY OF TECHNOLOGY. Downloaded on March 31, 2009 at 00:22 from IEEE Xplore. Restrictions apply.

```
200.     sim := sum / (queryNorm * wsdlNorm)
210.     r := new Result(wsdlId, score)
220.     rs.add(r)
230.   rs.sort()
240.   RETURN rs
```

The searching process consists of two parts – query vector formation and the cosine similarity calculation. The user query is first parsed into terms, each of which has the associated weights that constitute the original query vector components of Q^t (Line 60 and 80). Related term vectors in T are read (Line 10) from the term vector entity stored in the SVD index. Similarly, S is obtained from the singular value entity stored in the SVD index (Line 50). The query vector value is formed in Line 110. The final step of query vector formation is to calculate the norm of the query vector in order to compute the cosine. The second part, i.e. the cosine similarity, is to obtain the cosine angle between each WSDL vector and the query vector (Line 200). For the performance consideration, the WSDL vector norm is directly read from the SVD index since it is a predetermined constant (Line 170). The final result is sorted based on the cosine score in a descending order (Line 230). Moreover, only those WSDL documents whose similarity scores are greater than certain threshold are added to the final result list (Line 140 and 220).

5. Evaluation

In this section, we provide the evaluation result from our experiments in order to check whether the three requirements stated in the introduction section have been fulfilled by our prototype system – a Web services search engine.

Figure 5 shows a screenshot of our prototype system GUI – a typical search engine web page that displays a list of Web services on the topic "calculator", which has been input in the search box. Our LSA-based search engine has returned twenty Web services that can do certain kind of 'calculation'. Perusing the first service in the ranking list, (http://ausweb.scu.edu.au/aw02/papers/refereed/kelly/MathService.wsdl), we are unable to find any occurrences of string "calculator" in its WSDL file, or in its URL. However, this service is ranked at the first place when a service consumer needs a service such as a 'calculator'. Hence, LSA has automatically built a hidden semantic association between 'calculator' and 'maths' even though they do not co-occur in any WSDL files. Such a higher-order association cannot be captured by the VSM model with the original term-by-document matrix or any keyword-based service searching mechanisms.

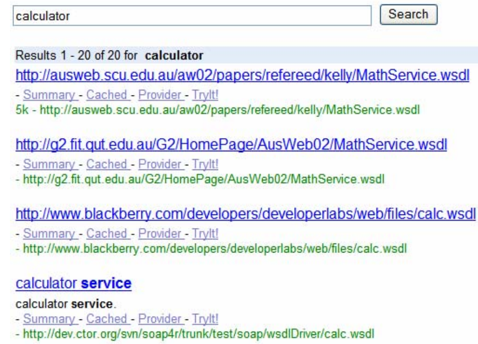

Figure 5 higher-order association between "Calculator" and "Math"

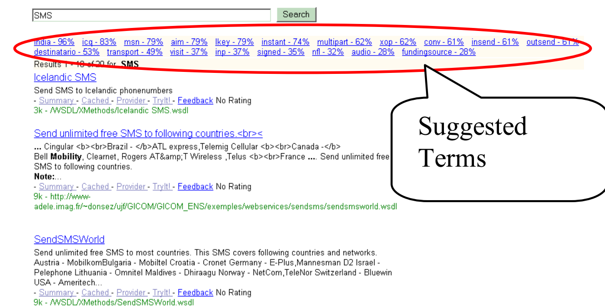

Figure 6 term suggestion as the primary use of higher-order association between terms

The primary use of higher-order association captured LSA is the term suggestion during the service searching. For example, a user who wanted to find Web services that can do things like "search" can be directed to find a Web services registry provided by the well-known UDDI vendor "Systinet" and its online Web services. Finding maths Web services, as another example, can be converted to find particular arithmetic operations such as multiply/substract/addition. The prototype in Figure 6 has shown that when the term "SMS" is typed, its associated twenty similar terms are suggested by the system. A user can then easily follow these terms to initiate another service searching request.

The higher-order associations between terms can also provide a cost-effective approach to build a light-weight ontology or taxonomy in a semi-automatic manner. An interesting research direction is thus to integrate these higher-order associations with end user activities such as feedback, blogging, and tagging to build and maintain a generic semantic space serving the user-centred semantic Web services retrieval and matching.

For the service clustering, we have employed the hierarchical clustering, particularly the Hierarchical Agglomerative Clustering (HAC) [30], to conduct the

Authorized licensed use limited to: CURTIN UNIVERSITY OF TECHNOLOGY. Downloaded on March 31, 2009 at 00:22 from IEEE Xplore. Restrictions apply.

clustering analysis. Hierarchical clustering is a widely used algorithm that works by grouping data into a tree of clusters. It has been reported in the literature [31-33] that hierarchical clustering algorithms generally yield better results than flat algorithms. HAC is a similarity-based bottom-up hierarchical clustering in which initially each WSDL document forms a cluster of its own as shown in the right part of Figure 7. HAC then iterates over the step that merges the two remaining most similar clusters into a larger cluster. The similarity measured here is referred to as the *combination similarity* between two clusters. A unique feature of HAC is that the higher the tree level, the lower the combination similarity is. As shown in Figure 7, at the leaf level of the tree, the combination similarity is "1", which assumes that each WSDL document is 100% similar to itself. With the mergers occur iteratively at higher levels, the similarity decreases from 80% to 60%, to 40%, until it reaches "0", where all WSDL documents are grouped into one cluster at the root level of the tree. The merging iteration can be terminated artificially at some desired combination similarity level (e.g. 0.2 in Figure 7) before all WSDL documents are grouped into one cluster.

The similarity measure for two WSDL documents is based on the inner product between two WSDL vectors with reduced number (i.e. 101) dimensions (i.e. factors) produced by SVD. For similarity between clusters (a group of WSDL documents), the similarity/distance measure is based on the average-linkage method [34], where the distance between two clusters is the mean of all pairwise distances between WSDL vectors contained in these two clusters. Figure 7 visualises the results for HAC-based service clustering using the Java Treeview[2], which includes three parts. The left most part is the hierarchical clusters generated from the HAC. The red circle represents a binary tree cluster where the lowest combination similarity is 0.24. Thus, the average similarities between all pairs of WSDL vectors are 24%. The middle part is the visualised semantic space, where the rows represent WSDL vectors and the columns stand for 101 factors (singular vectors). The red colour entry represents the positive value, and the blue one depicts the negative value. Entries with zero values are left blank. This middle part can be seen as a visualised representation of the raw data in **Error! Reference source not found.**. The right part illustrates the detailed Web services information in the corresponding cluster. The titles of these Web services provide data regarding the nature of this cluster – "*address*"-related Web services. The most unique part of service clustering is that it relies on the reduced semantic space A_k rather than the original vector space A. Therefore, semantically similar Web services are grouped together into hierarchical clusters.

[2] http://jtreeview.sourceforge.net

Figure 7 Hierarchical Agglomerative Clustering for WSDL corpus

6. Conclusion

Semantic-based service discovery requires: (1) semantic service searching, where the service meaning rather than texts are the focus of retrieval, (2) semantic term suggestion, where conceptually relevant suggestions are provided based upon service consumer's query terms, and (3) semantic service clusters, where semantically close Web services are grouped into hierarchical clusters for user friendly browsing and navigating. In this paper, LSA-based conceptual solution is proposed to deal with these three challenges. In particular, a five-step approach is presented to tackle the detailed technical issues, i.e. sparse matrix analysis and compression, singular value decomposition, cosine similarity calculation, term space construction, and hierarchical clustering. The prototype system of semantic-based service retrieval is developed and presented as a proof-of-concept system. For the future work, we would like to conduct extensive test in order to explore more quantitative experimental results for matchmaking performance.

7. References

[1] 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.

[2] D. Martin, "OWL-S: Semantic Markup for Web Services," in *Releases of DAML-S / OWL-S*, 2004.

[3] M. Paolucci, K. Sycara, T. Nishimura, and N. Srinivasan, "Using DAML-S for P2P Discovery," *Proceedings of First International Conference on Web Services, ICWS*, 2003.

Authorized licensed use limited to: CURTIN UNIVERSITY OF TECHNOLOGY. Downloaded on March 31, 2009 at 00:22 from IEEE Xplore. Restrictions apply.

[4] IBM and UGA, "Web Service Semantics," IBM and UGA 2005.
[5] M. Bruno, G. Canfora, M. Di Penta, and R. Scognamiglio, "An approach to support web services classification and annotation," presented at International Conference on e-Technology, e-Commerce and e-Service (EEE-05), 2005.
[6] C. Platzer and S. Dustdar, "A vector space search engine for Web services," presented at Third IEEE European Conference on Web Services, Sweden, 2005.
[7] M. Stal, "Web Services: Beyond Component-based Computing," *Communication of the ACM*, vol. 45, pp. 71 - 76, 2002.
[8] D. Karastoyanova and A. Buchmann, "COMPONENTS, MIDDLEWARE AND WEB SERVICES," Technische Universität Darmstadt, 2003.
[9] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web Services Description Language (WSDL) 1.1," 2001.
[10] A. Zaremski and J. Wing, "Signature Matching: A tool for using software libraries," *ACM Transactions on Software Engineering and Methodology*, vol. 4, pp. 146 - 170, 1995.
[11] G. C. Gannod and S. Bhatia, "Facilitating Automated Search for Web services," presented at IEEE International Conferences on Web Services, 2004.
[12] 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.
[13] A. Sajjanhar, J. Hou, and Y. Zhang, "Algorithm for Web Services Matching," presented at APWeb, 2004.
[14] 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.
[15] N. Kokash, "A Comparison of Web Service Interface Similarity Measures," University of Trento 2006.
[16] M. Y. Lin, R. Amor, and E. Tempero, "A Java reuse repository for Eclipse using LSI," presented at Australian Software Engineering Conference, 2006.
[17] Y. Ye, "Supporting component-based software development with active component retrieval systems," in *Computer Science*: University of Colorado, 2001.
[18] C. Wu and E. Chang, "Searching services “on the Web”: A public Web services discovery approach," presented at THE THIRD INTERNATIONAL CONFERENCE ON SIGNAL-IMAGE TECHNOLOGY & INTERNET–BASED SYSTEMS, Shanghai, China, 2007.
[19] G. K. Zipf, *Selected Studies of the Principle of Relative Frequency in Language*. Cambridge, MA.: Harvard University Press, 1932.
[20] S. C. Oh, H. Kil, D. Lee, and S. R. T. Kumara, "WSBen: A Web Services Discovery and Composition Benchmark," presented at IEEE International Conference on Web Services, 2006.
[21] J. Fan and S. Kambhampati, "A Snapshot of Public Web Services," *ACM SIGMOD Record*, vol. 34, pp. 24 - 32, 2005.
[22] I. Duff, R. Grimes, and J. Lewis, "Sparse Matrix Test Problems," *ACM Transactions on Mathematical Software*, vol. 15, pp. 1-14, 1989.
[23] M. W. Berry, "Large scale singular value computations," *International Journal of Supercomputer Applications*, vol. 6, pp. 13 - 49, 1992.
[24] J. K. Cullum and R. A. Willoughby, *Lanczos Algorithm for Large Symmetric Eigenvalue Computations*, vol. 1. Boston: Birkhauser, 1985.
[25] J. L. Goldberg, *Matrix Theory with Applications*. New York: McGraw-Hill, Inc., 1992.
[26] D. M. Berry, T. Do, G. W. O'Brien, V. Krishna, and S. Varadhan, "SVDPACKC (Version 1.0) User's Guide," Computer Science Department, Univeristy of Tennessee 1993.
[27] B. N. Parlett and D. S. Scott, "The Lanczos algorithm with selective reorthogonalization," *Math. Comp.*, vol. 33, pp. 217 - 238, 1979.
[28] J. Caron, "Experiments with LSA Scoring: Optimal Rank and Basis," Computer Science Department, University of Colorado at Boulder 2000.
[29] M. W. Berry, Z. Drmac, and E. R. Jessup, "Matrices, Vector Spaces, and Information Retrieval," *SIAM Review*, vol. 41, pp. 335 - 362, 1999.
[30] W. Day and H. Edelsbrunner, "Efficient algorithms for agglomerative hierarchical clustering methods," *Journal of Classification*, vol. 1, 1984.
[31] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. Englewood Cliffs, NJ: Prentice Hall, 1988.
[32] D. R. Cutting, J. O. Pedersen, D. Karger, and J. W. Tukey, "Scatter/gather: A cluster-based approach to browsing large document collections.," presented at ACM SIGIR 1992, 1992.
[33] B. Larsen and C. Aone, "Fast and effective textmining using lineartime document clustering," presented at The fifth ACM SIGKDD international conference on Knowledge discovery and data mining, New York, NY, USA, 1999.
[34] J. Han and M. Kamber, *Data Mining - Concepts and Techniques*: Academic Press, 2000.

Authorized licensed use limited to: CURTIN UNIVERSITY OF TECHNOLOGY. Downloaded on March 31, 2009 at 00:22 from IEEE Xplore. Restrictions apply.