A Specialized Search Engine for Web Service Discovery

Ourania Hatzi, Georgios Batistatos, Mara Nikolaidou, Dimosthenis Anagnostopoulos
Department of Informatics and Telematics
Harokopio University of Athens
Athens, Greece
{raniah, it20725, mara, dimosthe}@hua.gr

Abstract— **Web service discovery on the web is not a trivial task as the number of available web service descriptions continuously increases, and global UDDI registries are no longer available. As discovery through conventional, general-purpose search engines does not yield satisfactory results, a more promising alternative should be explored through specialized search engines. This paper explores the design and implementation of such a framework, resulting in WESS, a search engine targeted to discovering and retrieving web service descriptions. The presented system features an adaptive web service description collection process, through specialized and directed crawling, as well as an enhanced indexing and retrieval mechanism, which handles description documents as semi-structured text, separating actual information from tags and annotations. The paper also presents experiments and use cases regarding different search scenarios, in addition to performance results.**

Keywords- Web Service Discovery, Specialized Search Engine, Targeted Crawling.

I. INTRODUCTION

Web service discovery mechanisms are essential in order to exploit functionality offered on the web as a web service. Moreover, automation of the discovery process significantly facilitates the use of web services as reusable software components, which can be combined and composed to implement intricate procedures, through web service composition mechanisms.

Web service discovery was initially intended to be performed through Universal Description Discovery and Integration (UDDI) registries [24]. UDDIs started as global registries which would concentrate all available web service descriptions, represent them in a typical way, provide categorization in the form of white and yellow pages, and facilitate locating the appropriate web services. However, the vision of such global registries was abandoned; instead, the UDDI standard is currently used only for private registries.

A solution regarding discovery of web services exposing functionality on the web are conventional means of information discovery, namely search engines [7]. Search engines are designed to deal with significantly large volumes of data; therefore, they can handle the issue of the increasing number of available web services. However, general-purpose search engines are based on keyword search, taking into account the full text of the documents they index. As far as web service descriptions are concerned, indexing the full document and performing simple keyword search does not yield optimal results, as tags used in the document to annotate web service description elements appear multiple times, thus gain more importance than essential information, and skew search ranking and results.

This paper presents the design and development of WESS (Web Service Search Engine), a specialized search engine targeted to discovering web service descriptions on the web, both semantic (Web Ontology Language for Services - OWL-S, Semantic Annotations for WSDL - SAWSDL) and non-semantic (Web Service Definition Language - WSDL). The proposed approach maintains the significant features of general purpose search engines, such as scalability and dynamic content retrieval, while at the same time adapts and applies techniques for optimizing collection, indexing and retrieval of web service descriptions. The proposed search engine framework is modular; therefore, this approach can serve as a paradigm of targeted search for other cases of semi-structured text.

The rest of the paper is structured as follows: Section II presents related work in the area of search engines, both general purpose and web service specific. Section III provides an overview of the proposed architecture, while Sections IV to VI elaborate on the components of the system. Section VII presents use cases and results. Finally, Section VIII concludes the paper and poses future directions.

II. RELATED WORK

General purpose search engines, such as Google [10], Bing [2] or Yahoo [25], among other documents, index web service descriptions. However, searching for web service descriptions by querying such search engines has many disadvantages:

- The typical number of retrieved results is significantly large; however, only a small fraction of them concerns web service descriptions, even after adding appropriate keywords to the query; indicatively, experimental results show that only about 12% of the returned results are accompanied by typical WSDL descriptions [13].
- Typical search engines perform lexical analysis of all terms in each document. However; this approach is not appropriate for semi-structured text, containing annotations and tags, which is the case with web service descriptions, as each tag may appear multiple times in a document. Typical ranking algorithms

annotate this term with an increased weight, skewing search results.

- In typical search engines, users cannot define search parameters specific to web service descriptions; that is, they cannot indicate the part of the web service description in which specific keywords should be located. Moreover, general purpose search engines do not permit search for non-functional service requirements, such as quality of service, since such requirements are defined not by keywords but by key-value pairs.

Filtering results obtained from general purpose search engines to keep only web service descriptions would be an intermediate solution; however, such an approach would not be efficient and as effective as a specialized search.

For these reasons, specific purpose search engines targeting web service discovery have been proposed. In these approaches, web service descriptions are collected either through UDDI registries, where providers are required to manually list their web services by registering their descriptions, or by crawling the web; search is usually performed by keyword queries.

Woogle [6] acquires web service descriptions through UDDI registries. Search is performed through keywords, and offers the possibility to discriminate between inputs and outputs. Woogle also employs a clustering mechanism, enabling the user to describe a desired web service functionality model and receive web services that fit that model. Finally, Woogle offers the possibility to satisfy user requirements by suggesting a composite web service, consisting of a combination of available atomic web services. Currently, no running implementation of Woogle can be found.

VUSE [17] also uses UDDI registries to obtain web service descriptions. Web service descriptions are represented as vectors and a vector space model along with the TF-IDF metric is utilized for ranking similarity between them. VUSE is another example of specialized web service search engine that does not offer an available implementation.

Seekda [22] utilizes the latest web service technology to provide business solutions, and to facilitate that it offers a search engine specialized for web services. Seekda employs the generic Heritrix Web Crawler [12] to discover web services, using as seeds webpages discovered by the general-purpose search engine Alexa. The web page descriptions are consequently processed and their terms are annotated with weights. So far, Seekda indexes approximately 28000 web services from 7500 providers. Search is performed through keywords and logical operators. Results are ranked not only according to similarity to the query, but service response time and service reliability are also taken into account.

WSExpress [26] crawls the web in order to find web service descriptions and indexes them using both functional and non-functional characteristics. Non-functional evaluation is performed through experiments, while for functional characteristics the TF-IDF model is utilized. Search ranks the results taking into account both types of characteristics.

SWSE [14] and OSSE [5] attempt to increase recall and precision in web service search by incorporating semantics retrieved from ontologies, for OWL-S web services. However, neither work provides a solid implementation or experimental data in order to further evaluate its results.

Popular web service registries focus on enhancing web service retrieval, based on a UDDI registry, in an attempt to increase precision and recall, disregarding the process of web service collection. However, UDDI registries, where a web service provider registers available web services, diminish. Since the number of web service descriptions available on the web but not listed in a UDDI registry constantly increases, building and maintaining a dynamic available web service repository is very significant. Only a few web service search engines crawl the web for discovering web service descriptions; however, crawling is not targeted, therefore collected results are discouraging [22]. Furthermore, most web service search models employ the TF-IDF ranking scheme [1], without adjusting it to the web service case that discriminates between different parts of web service description documents. Finally, most of the aforementioned approaches are research efforts that do not result in solid tools that could be used to draw useful conclusions.

The motivation of the work presented in this paper stems from the requirement for a solid web service discovery mechanism with a high degree of automation, purposefully designed to accommodate current discovery issues and specifically adapted to the web service case. UDDI registries and general purpose search engines cannot cover this requirement, while aforementioned specific search engine approaches present shortcomings, as discussed above. Thus, we explored the development of a web service search engine from scratch. The framework presented in this paper is intended to serve as a dynamic repository for available web services, which is continuously updated, based on available web service descriptions, and automatically maintained; therefore, it can capture and accommodate the dynamic environment of the web. Apart from web service discovery purposes, such a repository is intended to be incorporated as a component in the PORSCE web service composition framework [11]; in this case. A dynamic repository would significantly facilitate composition, as it would provide a variety of available web services as building blocks and it would also enable the location of semantically equivalent or similar services when semantically relaxed composition is required.

III. WESS ARCHITECTURE

The work presented in this paper concerns the design and development of WESS, a search engine specific for web service descriptions, both semantic (OWL-S and SAWSDL) and non-semantic (WSDL). The proposed system collects web service descriptions through targeted crawling, indexes them and retrieves them through advanced keyword search. For semantic web service descriptions, the corresponding ontologies are also indexed and retrieved. Currently, RESTful web services were not considered, as they do not comply to a description standard, which makes their identification and indexing difficult.

The proposed framework architecture is modular; therefore, the subsystems comprising it can be handled as individual independent entities, communicating with each other through standard languages or structured intermediate XML (Extensible Markup Language) files. Modularity enables the substitution of each component without the requirement to alter any of the other components, in order to experiment with different configurations and promote reusability.

WESS consists of a targeted web crawler, the WSDL/SAWSDL and OWL-S parsers, the indexing mechanism, taking into account the different parts of web service description documents, while a web interface is provided for users. Exchange of data among the components is performed through WSDL, OWL-S or XML custom files. The architecture is presented in Figure 1.

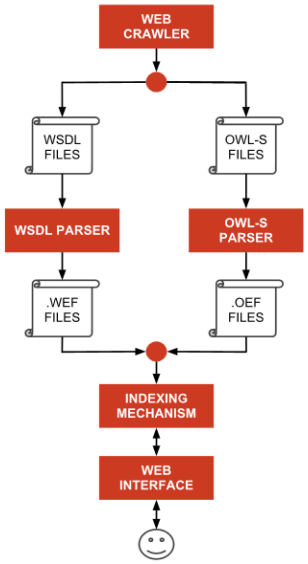

Figure 1. Proposed framework architecture.

IV. CRAWLER

Conventional web crawlers aim at exploring the largest possible portion of the World Wide Web, and index as much information as possible [3]. However, a specialized crawler for web service descriptions aims at exploring the web in such a way that it locates and indexes as many web service descriptions as possible, in a relatively short time. Therefore, the specialized web crawler does not index the entire amount of information it accesses on the web; it rather treats web pages as collections of links that allow further crawling [8].

Specialized web crawling handles the World Wide Web as a directed graph, much like conventional crawling, where web documents are nodes and links are directed edges between them. Known and established search algorithms can consequently be applied, featuring the search frontier, containing all web addresses that the crawler intends to visit, and the closed set, containing all web addresses that the crawler has already visited [23].

Existing crawlers, such as Heritrix [12], could be modified to perform specialized crawling; however, in the WESS case, the modifications would be heavy, while it was essential that the crawler could be deployed on a low-end machine; therefore, it was decided that a new crawler would be developed.

In the remainder of this Section, the basic functionality as well as extensions, adaptations and optimizations performed to the WESS crawler are presented. Among the goals of the specialized crawler that are pursued with these optimizations is the ability to perform well on low-end machines.

A. *Basic Functionality*

As far as the basic functionality is concerned, it is similar to that of a general-purpose web crawler. The specialized web crawler starts from an initial URL (Universal Resource Locator), forms an HTTP (HyperText Transfer Protocol) request and retrieves the corresponding document. It consequently checks whether the document is a web service description; in that case, the document is stored for future indexing. In any other case, the links contained in the document are retrieved and added to the search frontier, while the URL itself is added to the closed set. Search continues by visiting the first URL of the search frontier, confirming that is it not a member of the closed set, and repeating the procedure until the search frontier is empty. In general, new URLs retrieved from web pages are added each time to the end of the search frontier; therefore, Breadth-First Search (BFS) [20] is performed.

The initial set of URLs (seed set) included known service registries, such as the OASIS or IBM ones, as well as several web service catalogs retrieved by manual Google search.

B. *Multithreading*

In order to increase crawling speed, the basic functionality presented above was transformed into a multithreaded implementation [9]. The basic algorithm is divided into two discrete modules:

- management and interaction with the search frontier and the closed set in order to acquire the next URL, which must be performed in a centralized, synchronized manner, to ensure that the same URL will not be assigned to two different threads
- retrieval and processing of the web document, which is time-consuming and requires a lot of resources.

Multithreaded implementation assigns the process of retrieval and processing of each web document to a different thread. Multithreading significantly accelerates the crawling process; the maximum number of threads achieving optimal performance can be determined experimentally. Experimental performance results for the multithreaded implementation are provided in Section VII.A.

C. *Web service descriptions identification*

In order to detect whether a retrieved web document is a web service description, the following three methods have been utilized:

- Checking the extension of the retrieved document; web service descriptions of interest end in "`.wsdl`", "`?WSDL`", "`.owl`" or "`.owls`".

- Checking the "`content-type`" field of the HTTP response for web service descriptions MIME types.
- Checking the header of the source code of the retrieved web page for web service description indicators.

In order to successfully identify a web document as a web service description, at least two of the aforementioned detection methods have to provide positive results, since checking the content-type HTTP response field can yield false positives.

D. File avoidance mechanism

During search, the specialized web crawler came across web documents containing types of files that impeded search. Such files did not contain web service descriptions, did not have any links that could be used to continue search, and in the majority of cases utilized significant amounts of available memory and bandwidth. Examples of such files are image, sound and video files, applications and compressed archives. A typical search engine would be required to index metadata about such files along with their addresses, in order to make them retrievable; however, in the case of web service description search, they should be avoided in order to increase crawling performance. Using the same methods for detecting web service descriptions, i.e. checking extensions and the HTTP response header "content-type" field, these file types were identified and excluded from crawling.

E. Fragmentation of Search Frontier into files and Closed Set size limitation

An important issue that needed to be addressed by adaptations to the crawler was the minimization of the memory used for storing the search frontier and the closed set, as the size of these two structures continuously increases. In fact, as crawling speed increases, the issue becomes more intense; for each retrieved webpage, the retrieved links are on average more than one; therefore, the size of the search frontier increases exponentially. In order to cope with this issue, the way the algorithm uses the search frontier was examined. It was observed that the search frontier is an ordered list and the algorithm interacts only with its beginning, retrieving the URL to be examined, and its end, adding the URLs that were retrieved from the webpage. Based on these observations, the search frontier was divided into parts of specific size; at any time, only the first (head) and last (tail) parts are required to be in memory; the intermediate parts are written into files stored on disk. As the algorithm proceeded, if the head became empty, the next part was retrieved from the disk; if the tail reached a predetermined size, it was forwarded to the disk and a new tail list started. The files on disk were further compressed in order to decrease storage space.

As far as the closed set is concerned, the technique used for the search frontier does not apply in this case. Each URL has to be checked against all URLs contained in the closed set, and this process takes place very often; therefore, if parts of the closed set were stored on disk, the required disk accesses would degrade performance. A viable solution is to confine the closed set only to the more recent URLs examined, which prevents the algorithm from entering local loops. Re-visit of URLs is not fully prevented with this method; however, occasional re-visit could be useful for keeping the repository up-to-date.

F. Domain Ignore List Heuristic

Crawler performance is increased not only through crawling speed, but also by directing the crawler away from web domains that are either especially designed to hinder crawling (spider traps), or are generally expected to not contain web service descriptions. Such domains include large e-shops, such as amazon.com, or forums. This is implemented through a heuristic rule incorporated in the crawling algorithm, which intuitively defines that if the crawler visits a large number of pages from a specific domain, having located no web service descriptions among them, then the rest of the domain webpages can be safely ignored. Such domains are added to the domain ignore list, which is used to filter new URLs before adding them to the search frontier.

G. OWL-S to WSDL Heuristic

Since OWL-S Profiles do not contain concrete grounding information, it was observed that in many cases WSDL documents accompany OWL-S descriptions. In order to guide the crawler into locating these descriptions, a heuristic rule was developed, stating that whenever an OWL-S profile instance is found, the crawler will add a similar URL for a WSDL description to the search frontier as well.

H. Breadth to Depth Heuristic

Another heuristic rule incorporated in the crawler functionality dictates the local change of the crawling algorithm from breadth-first, which is the default search algorithm, to depth-first search, whenever a web service description is located, based on the observation that a provider often offers more than one web service. According to this heuristic rule, whenever the crawler locates a web service description, it performs a local depth-first search to the domain at hand, overriding the search frontier ordering. The application of this heuristic ensures that if the specific webpage offers links to more than one web service, their descriptions will be located faster.

V. INDEXING & RETRIEVAL

Currently, web service indexing and retrieval in WESS is performed using classical text retrieval methods, modified to suit the web service case as explained in the following. In the future, as semantic web services gain momentum and the crawler collects more semantic descriptions, alternative suitable methods, such as the OWLS-MX [15] or SAWSDL-MX [21] matchmakers could be incorporated in the retrieval module.

Collected web service descriptions are processed by the parsers, which execute a sieve-like job, extracting the suitable data from the descriptions into intermediate .wef and .oef XML files for WSDL/SAWSDL and OWL-S descriptions respectively.

The elements indexed for each web service description standard are the following:

- WSDL: *names, documentation, inputs, outputs*

SAWSDL: *names, documentation, inputs, outputs, model references (in the case of model references, the corresponding ontology elements could also be indexed, as the crawler stores the accompanying ontologies as well)*

- OWL-S: *service name, description, inputs, preconditions, outputs, results*

These data summarize the functionality exposed by the collected web services and thus, will be used to respond to user generated queries searching for specific functionality. To achieve this in a fast and accurate way they have to be stored in a data structure built solely for this purpose, i.e., a specialized index.

The index operation starts by reading and then parsing the intermediate files, followed by another round of data processing, called normalization, before indexing the data. In order to minimize index fragmentation, the data have to withstand a number of transformations beforehand. This means removing the contained numbers and punctuation, splitting camelCase tokens, turning all tokens to a single case, removing stop words and extracting the term stems using the Porter stemmer for the English language [18].

Subsequently, the index is built, corresponding to a hash table where the web service description id is the key leading to the web service contained tokens. However, responding to a query requires the opposite process; that is, locating the descriptions that contain the query terms quickly before ranking those descriptions. Hence, an inverted index – again a hash table – is created using the existing index, where the hash key is the term leading to all the descriptions that contain it [4]. By now, the web service descriptions can be regarded as vectors in accordance to the vector space model, allowing their interpretation with a number of different methods [16].

During the index formation, a number of measures are calculated, namely the term frequency (TF) and inverse document frequency (IDF), for each term contained in the index, according to the classical information retrieval model TF-IDF. The above measures are then used to calculate the cosine similarity between the user generated query, depicted as another description, and the descriptions that contain its terms.

It must be noted that there are two types of user generated queries with the first one applying to the default conventional search through the TF-IDF model, and the second one applying to the advanced search where the user can specify the part of the web service description where a query term has to appear. In the last case, an additional weight is introduced in the TF-IDF as shown below. These measures are calculated as following [1]:

$TF = \frac{f}{l}$, where f is the frequency of the term in the description normalized by the description length l and $IDF = \log(\frac{|D|}{1+d})$, where D is the total number of web service descriptions and d the number of descriptions containing the term. Finally, $TFIDF = TF \cdot IDF$, in the simple search case while $TFIDF_{ws} = TF \cdot IDF \cdot weight$, for the advanced search case.

The similarity between a user query and a web service description, both represented in the TF-IDF model, is calculated as $\cos\theta = \frac{d \bullet q}{\|d\|\|q\|}$, where d is the description and q is the query, both expressed as vectors. The cosine similarity model was selected in the current implementation of WESS due to its close relation with the TD-IDF weighting scheme and the simplicity of its implementation.

VI. Interface

User interaction is performed by means of a web interface front end, which communicates with the indexing back end utilizing the web service technology. This ensures independence between the two components; while, the search engine back end can be incorporated in other systems as well, as a web service component.

The web interface offers the possibility of simple and advanced search, through keywords. Simple search functionality includes content retrieval based on generic keyword matching, much like conventional search engines [19]. It can be used in cases when the user wishes to locate web services regarding a specific domain. Advanced search is specialized for web service descriptions, taking into account not only keywords, but additional web service characteristics as well. Advanced search enables users to request for keywords in a specific part of the web service description, such as specific inputs and/or outputs. In that way, they are enabled to search for desired functionality, rather than information.

VII. Experiments, Results and Use Cases

A. Crawler Performance

The web service search engine was installed on a workstation at the Department of Informatics and Telematics, Harokopio University of Athens. During the first two months of the crawler operation, over 22 million links and 9.3 million different webpages have been examined, locating approximately 1500 different web service descriptions, 97% of which concerned WSDL descriptions, and the rest OWL-S and SAWSDL. The initial number of retrieved webpages per day was approximately 100.000, while, after the aforementioned optimizations such as multithreading, the file avoidance mechanism and the domain ignore list heuristic, this number reached 1.5 million webpages; revealing satisfactory performance results for execution on a typical low-end workstation.

As far as the optimal number of threads is concerned, there is a trade-off between increase in crawling speed and synchronization overhead. More specifically, it was observed that performance reaches a plateau within a specific range of number of threads; an optimal number of threads yielding minimum crawling time can be found within this range. Too few threads do not fully exploit the capabilities of the available processing and networking infrastructure, resulting in significantly decreased performance. On the other hand, if

the number of threads increases excessively, performance degrades dramatically, as synchronization between the threads is a bottleneck point. Controlled experiments were performed in order to determine the optimal number of threads for the specific configuration (processing power, memory, available bandwidth). Experiments measured the time required to crawl a test set containing 10.000 URLs, as the number of crawler threads increased, yielding a plateau between 600 and 2100 threads and an optimal number of approximately 1850 threads, as shown in Figure 2.

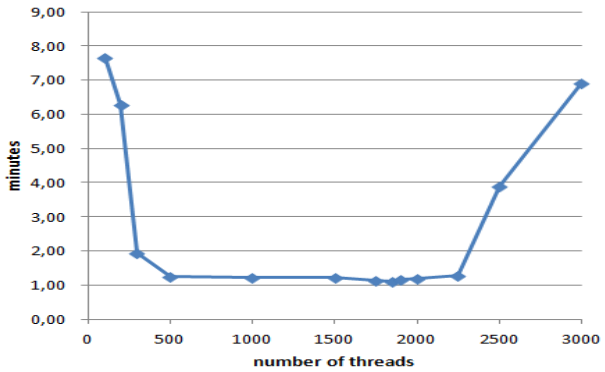

Figure 2. Crawling time with number of threads.

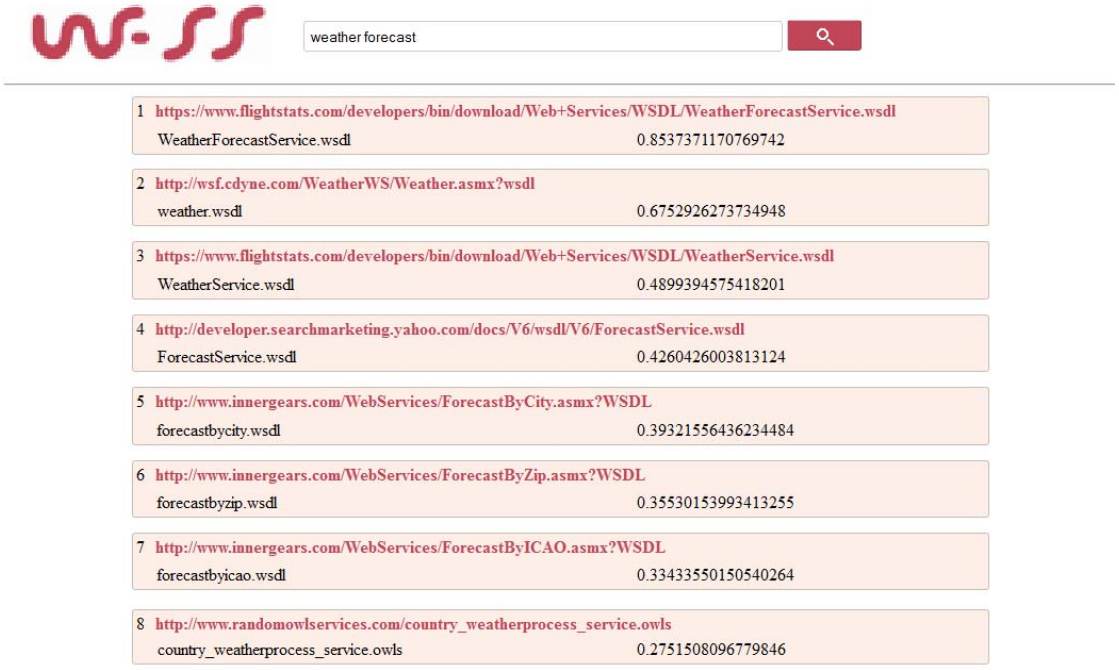

Figure 3. Top results for search with keywords “weather forecast”.

B. *Use Cases*

If a user wishes to find web services regarding the weather forecast domain, a keyword search produces the results depicted in Figure 3. The result list contains descriptions both in the WSDL and OWL-S standards, as both of these are indexed by the search engine, while each result is accompanied by its score, which occurs by the adjusted TF-IDF scheme described in Section V. Note that the score of the “WeatherForecastService.wsdl” is significantly higher than the rest, since both keywords are found in the name of the web service, as opposed to other descriptions, whose name contains only one of the two keywords.

Simple search follows the conventional standards of retrieval through keyword matching, adjusted to the web service descriptions case. The benefits of WESS in this case, as opposed to general purpose search engines, are that results include only web service description documents, while tags annotating web service description elements are not taken into account. A comparison with specialized search engine approaches was not possible, as Seekda does not use the same retrieval model, while for the rest we were unable to locate functional implementations.

Advanced search offers the possibility to search for functionality, as opposed to search for information. Advanced search is accommodated by describing functional web service requirements, that is, inputs and outputs, as also suggested in Woogle. Unfortunately, we were unable to identify the differences between the two approaches as currently no running implementation of Woogle was found.

In order to illustrate the differences between the two types of search, for the same search terms, in the following, an example is provided. Figure 4 shows the top results of simple search using the keyword “weather”, which yields all available web service descriptions that contain this term in

any part of them. Figure 5 depicts the top results of advanced search with the same keyword; advanced preferences denoted that the term "weather" should appear only in the outputs (results) of the web service. Advanced search returns significantly fewer results, as web service descriptions that contain the keyword in any other part of the description other than the output element are excluded. Moreover, the similarity measure value for each web service description is different.

In contrast to other approaches, the proposed framework collects the web service description documents by crawling the web; therefore it includes descriptions not listed in UDDI registries; moreover, the crawling process is targeted to web service descriptions, through heuristic rules. At the same time, although indexing is based on the established TD-IDF model, it takes into account features of web service descriptions and adapts the model accordingly, providing the ability to search for specific web service functional characteristics.

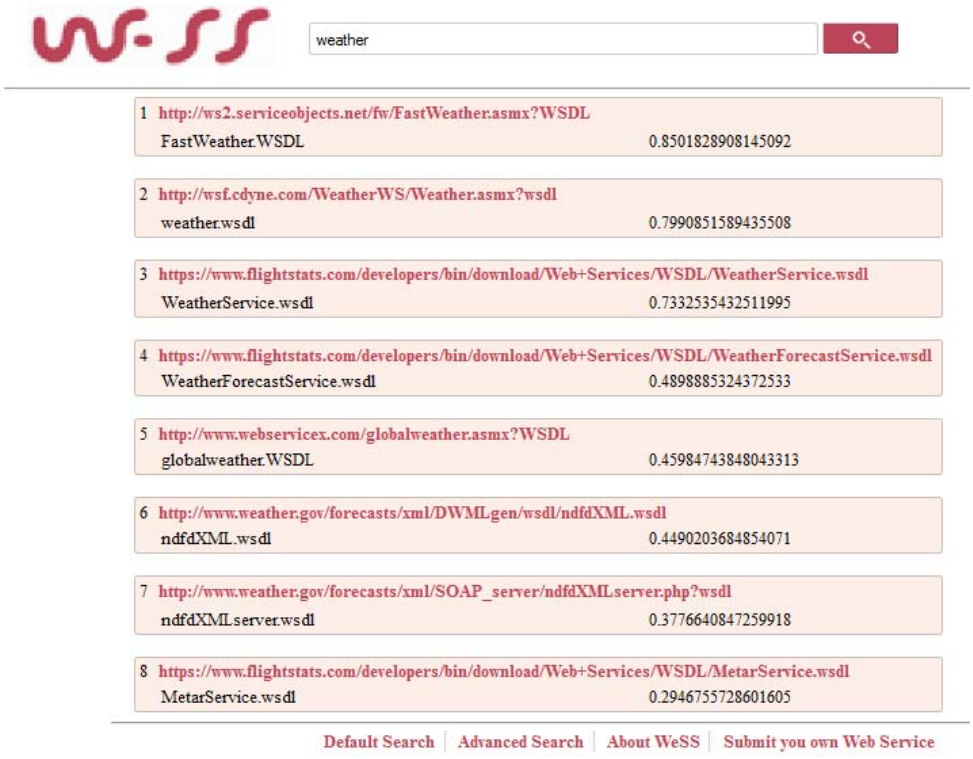

Figure 4. Top results for simple search with keyword "weather".

Figure 5. Results for advanced search with output "weather".

VIII. Conclusions & Future Work

Web service discovery on the web is a complex and time-consuming task; global UDDI registries are not available, while locating web services through general-purpose search engines is hardly a solution. This paper tackles this issue through the design and implementation of a web service specific search engine that aims at significantly facilitating the automation of web service discovery and composition.

The proposed framework is modular, in order to serve reusability and experimentation with different subcomponents. The framework includes a web crawler that

performs targeted crawling on the web, collecting available web service descriptions in a selective and directed manner. In addition, an indexing mechanism is included, which extends and adapts the TF-IDF model to the web service case. The querying mechanism is based on keyword search; however, complex expressions, including not only logical operators but also structural information regarding the web service descriptions, are employed. Finally, the proposed web service specific search engine offers a web interface, while inter-component communication is performed utilizing the web services technology.

Experiments yielded promising results, as targeted web crawling directed web search so that more available web service descriptions were located faster than by regular crawling. Moreover, modifications to the crawling algorithm enabled its efficient execution even on low-end machines.

The proposed indexing and search mechanisms were extended and adapted. Their significant characteristic is that keyword search is not performed throughout the entire web service description text; instead, descriptions are partitioned into their structural elements, such as inputs and outputs, and search is performed taking into account the description part where each search term is found. As experiments confirm, the adapted indexing and search mechanisms yield optimized results as far as ranking of the returned web service descriptions are concerned, as, during indexing, terms denoting tags are disregarded.

Our experience from exploiting the proposed framework revealed that currently, the majority of web service descriptions publicly available on the web concern the WSDL standard, and only a few are expressed in semantic languages, i.e. SAWSDL and OWL-S. Although such an observation can be explained due to the automatic creation of WSDL descriptions through development tools, the web service community needs to address the semantics issue, in order to facilitate web service related procedures on the semantic level. As the notion of Semantic Web gains momentum, web service discovery and composition approaches should take into account semantics; the proposed approach is designed to handle both semantic and non-semantic web service descriptions while crawling and indexing.

The proposed framework serves as a paradigm of specialized search and information retrieval, which could be applied to other cases of semi-structured text as well, with minimal intervention. Required modifications include changes to the crawling algorithm, depending on the distribution of the documents of interest on the web, as well as the assignment of different combination of weights to the different text parts of the indexed documents.

Future directions also include the enhancement of the proposed framework with semantic information from a variety of sources, such as ontologies, lexical databases or thesauri. Such enhancement would enable semantic search, and would also facilitate approximations, in cases when exact results cannot be found. Moreover, additional techniques for retrieval of web service descriptions should be explored. Finally, we intend to explore heuristics for identifying and indexing RESTful web service descriptions.

REFERENCES

[1] Baeza-Yates, R., Ribeiro-Neto, B., Modern Information Retrieval. New York: ACM Press, Addison-Wesley, 1999.

[2] Bing, http://www.bing.com/ [accessed 06/02/12]

[3] Castillo, C., Effective Web Crawling, University of Chile, 2004.

[4] Cutting, D., Pedersen, J., Optimizations for dynamic inverted index maintenance, Proceedings of SIGIR, 405-411, 1990.

[5] Dan, G., New ideas for Web service discovery-ontology-based prototype system of service search engine, 2nd International Conference on Software Technology and Engineering, 2010.

[6] Dong, X., Halevy, A., Madhavan, J., Nemes, E., Zhang. J., Similarity search for web services, Proceedings of VLDB, 2004.

[7] Doulas, A., Chv, J., Olston, C., What's new on the web?: the evolution of the web from a search engine perspective, Proceedings of the 13th international conference on World Wide Web, ACM New York, NY, USA, 2004.

[8] Edwards, J., McCurley, K. S., and Tomlin, J. A., An adaptive model for optimizing performance of an incremental web crawler, In Proceedings of the Tenth Conference on World Wide Web (Hong Kong: Elsevier Science), 2001.

[9] Goetz et al, Java Concurrency in Practice, Addison Wesley Professional, 2006.

[10] Google, http://www.google.com/ [accessed 06/02/12]

[11] Hatzi, O., Vrakas, D., Nikolaidou, M., Bassiliades, N., Anagnostopoulos, D., Vlahavas, I., "An Integrated Approach to Automated Semantic Web Service Composition through Planning", IEEE Transactions on Services Computing, 2011.

[12] Heritrix Web Crawler, http://crawler.archive.org/ [accessed 06/02/12]

[13] Lausen , H., Haselwanter, T. Finding Web Services, Proceedings of the 1st European Semantic Technology Conference, 2007.

[14] Ma, C., Song, M., Xu, K., Zhang, X., Web Service discovery research and implementation based on semantic search engine, 2nd Symposium on Web Society, 2010.

[15] OWLS-MX Web Service Matchmaker, SemWebCentral, http://www.semwebcentral.org/projects/owls-mx/ [accessed 29/04/12]

[16] Paolucci, M., Kawmura, M., Payne, T., Sycara, K., Semantic Matching of Web Services Capabilities, First International Semantic Web Conference, 2002.

[17] Platzer, C., Dustdar, S., A Vector Space Search Engine for Web Services, In Proceedings of the 3rd European IEEE Conference on Web Services, 2005.

[18] van Rijsbergen, C.J., Robertson, S.E., Porter, M.F., 1980. New models in probabilistic information retrieval. London: British Library. (British Library Research and Development Report, no. 5587).

[19] Rose, D. E., Levinson, D., Understanding user goals in web search. Proceedings of the 13th international conference on World Wide Web, ACM New York, NY, USA, 2004.

[20] Russel, S., Norvig, P., Artificial Intelligence, A modern approach, 2003

[21] SAWSDL-MX Web Service Matchmaker, SemWebCentral, http://www.semwebcentral.org/projects/sawsdl-mx/ [accessed 29/04/12]

[22] Seekda!, Web Services Search Engine, http://webservices.seekda.com/ [accessed 06/02/12]

[23] Shkapenyuk, V., Suel, T., Design and implementation of a high performance distributed web crawler, In Proceedings of the 18th International Conference on Data Engineering (ICDE), pages 357-368, San Jose, California, IEEE CS Press, 2002.

[24] UDDI Specification version 3.0.2, http://www.uddi.org/pubs/uddi_v3.htm [accessed 06/02/12]

[25] Yahoo!, http://search.yahoo.com/ [accessed 06/02/12]

[26] Zhang, Y., Zheng, Z., Lyu, M.R., WSExpress: A QoS-aware Search Engine for Web Services, International Conference on Web Services (ICWS), 2010.