Enhancing Discovery of Web Services through Optimized Algorithms

Janette Hicks, Madhusudhan Govindaraju, Weiyi Meng
Department of Computer Science, State University of New York (SUNY) at Binghamton, NY
{jhicks, mgovinda, meng}@binghamton.edu

Abstract

This paper describes research in methods to discover Web Service Description Language (WSDL) documents. This work extends current discovery research through use of the Google Web service, UDDI category searching, and private registry querying with preliminary experiments resulting in 97% success. The goal is to find WSDL documents for a given domain name, parse the desired service document to obtain invocation formats, and automatically invoke the Web service. Contributions of this research will support enhancements of HTML-dependent search tools by providing access to data inaccessible through surface HTML interfaces.[1]

Keywords: Web service, UDDI, Search algorithms.

1. Introduction

The goal of this project is to enhance discovery of Web services through design of automated discovery algorithms to provide a foundation for utilization of Web services in enhancing meta-search engine capability. Web services have the potential to significantly enhance several aspects of meta-search engines. A known problem is incorporating search results from HTML search engine pages because the output must be sensibly and accurately parsed to be correctly utilized. In contrast, description of the service and details of invocation and response are well defined with Web services. Service specifications are defined in the WSDL document for a service via XML-Schema based type system. Paralleling discussions on tapping the "hidden web" [3], Web services have the functionality to provide access to information not currently available on the "surface" web, the web represented by HTML pages. While search engines are also more homogeneous in grouping data, Web services allow groupings of functionality that is heterogeneous in nature. For example, a meta Web service for travel-related Web services grouped into an overall plan including airfare, hotel, and rental car. Web services allow for expansion of existing search engine technology and access to greater depths of the Internet offered information while providing data in a highly structured format. This discovery research targets any kind of Web service that currently exists and is accessible from the Internet. Google and Amazon represent the broad range of search services for which Web service interfaces are currently available. This project includes search of any type of Web service, since *discovery* techniques should be applicable to future Web services developed for search engines.

In researching Web service discovery standards we found that other than those based on a central registry service, no standard is currently widely implemented. Proposed standards include Microsoft's Web Services Dynamic Discovery (WS-Discovery) [4], which is a dynamic discovery specification that defines a multicast-based ad-hoc protocol to find Web services on a network. An older Microsoft standard, Discovery of Web Services (DISCO) is a static standard using *.disco* documents for a domain to identify the WSDL documents [2]. The WS-Inspection Language (WSIL) [5] is a similar lightweight discovery standard promoted by Microsoft and IBM that merges their earlier efforts: IBM's Advertisement and Discovery of Services (ADS) and Microsoft's DISCO. WSIL documents contain locations of WSDL documents and pointers to other WSIL documents to create hierarchical lookups for services [1].

Another source of Web services is independent Web service "brokers" who feature their own registries of Web services, such as Xmethods at www.xmethods.com. The main source of discovery proposed in current research is the use of a universal public registry, such as the public Universal Description, Discovery, and Integration (UDDI) [UDDI 05] Universal Business Registry (UBR) jointly hosted by Microsoft, IBM, SAP, and NTT Communications.

The research question addressed in this paper is how to determine if a given domain or web site provides Web services interface(s) and, if so, derive the WSDL document location(s). The search includes current

[1] This work is supported in part by NSF grants IIS-0414981 and CNS-0454298.

0-7695-3032-X/07 $25.00 © 2007 IEEE
DOI 10.1109/GrC.2007.146

discovery methods and new algorithms we have developed for this purpose. Once the desired WSDL document is found, services and operations are parsed and presented to the user who decides on options for dynamic invocation of the service. The project design and implementation are set up to start with a given domain name and to measure, using developed methods, how well a specific WSDL document is successfully found for this domain across all Internet resources. Current standards only cover discovery of WSDL documents in a central registry or propose standards that are not universally implemented, resulting in a lack of coverage of all existing WSDL documents.

An initial pool of 40 WSDL documents was used in developing these discovery techniques. Final experiments using a test base of 80 WSDL documents produced results with over 97% success rates in finding those documents using these methods. In developing techniques to achieve this success, numerous experiments were conducted, with minor variations in search terms used, types of queries, and sources.

2. Web Service Search Algorithms

We have developed several discovery methods including use of the UDDI public registry at Microsoft to produce a final result of "found" or "not found."

2.1 Method One – Google WSDL Search

Our first approach is use of the Google web service API and developer kit to use the Google search engine for WSDL references. Four different options are:
(1) Stemmed Terms Google Search - uses stemmed terms from the home page.
(2) Unstemmed Terms Google Search - uses unstemmed terms from the home page. Option 1 and 2 incorporate parsing of HTML pages including stop word removal and stemming, term frequencies, and other parsing utilities. This functionality refines terms to search using APIs with use of <title>, <meta>, and other HTML tag terms as well as high frequency terms.
(3) Google "inurl" Keyword Search - uses keyword "inurl:" with the domain term (i.e., if the URL entered was www.gotdotnet.com, the domain term is "gotdotnet", which the software parses from the URL input.) So, the parameter is "inurl:gotdotnet". The parameter "inurl:wsdl" is also added.
(4) Google "site" Keyword Search - uses keyword "site:" and the URL input – for example: "site:www.gotdotnet.com" – along with "inurl:wsdl". In our experiments, the latter two methods proved superior to the first two options.

2.2 Method Two – Google WSIL search

This approach uses the Google web service APIs and developer kit to search Google for WSIL references. This process is directed at finding a WS-Inspection document that can be parsed for WSDL document locations. Two options exist:
Google "inurl:" Keyword Search – uses "inurl:" with the domain term. So the parameter used is "inurl:gotdotnet" along with "inurl:wsil" for the domain *gotdotnet.*
Google "site:" Keyword Search – uses "site:" and the URL as input – for example: parameter "site:www.gotdotnet.com" along with parameter "inurl:wsil". The results of these searches are stored in a list for later processing.

2.3 Method Three – Crawl WSDL Search

Method three uses web crawling to try and locate a WSDL document for a domain. Crawling starts at the home page (input to the program), and goes down a specified number of levels. Two crawlers are used by our toolkit: 1) WebSPHINX, which is a complex crawler configurable with a GUI front end and 2) a simpler crawler developed for this project. The WebSPHINX crawler engine (separated from the GUI) is implemented with variations on methods to try and identify WSDL documents and direct the crawler. It includes options such as choosing to stay within the domain or (2) only crawl pages that have URLs with the key term of the domain. The simple crawler program is naïve, which resulted in inconsistencies in our runs. So for our experiments we also used the Google web service with the "site:" parameter to activate the Google crawler.

2.4 Method Four – UDDI Query

Method four is a search implemented using the Sun Java Web Services Development kit (JAXR) to query the registry. UDDI registry queries by Name (organization) and Category (tModel) are incorporated to try and find WSDL documents within the Microsoft development registry. Query by name uses the domain of the input home page and tries to find a matching business name. From there the tModel entry can be read in order to find the WSDL document location. Category search involves searching through all tModel entries in the registry with classification "wsdlspec" and searching for the domain in the list of WSDL document names.

During our experiments using the Microsoft development registry, we discovered that only the first 1000 results were returned for a query, with no method to get the "next 1000". Our code was therefore modified to retrieve tModels in smaller alphabetical groups, which greatly increases searching time. In the production UDDI, which has many more entries, it would not be feasible to use this approach for a dynamic search and invocation scheme. Also, manually searching the same registry using a tokenized version of the home page yields more satisfactory results. If the domain name could be tokenized then the WSDL document could be more reliably found.

2.5 Method Five: Use StrikeIron Registry

Method five is an example of searching a registry not associated with the public Universal Business Registry (UBR), such as the Microsoft UDDI registry. In this case, the StrikeIron registry was used. This registry is currently not automatically maintained, so Web services are not dynamically discovered. Instead, service providers register their services in a maintenance manner similar to the UBR. StrikeIron has its own software and web interface to search this registry. In searching, a URL is passed to StrikeIron, which simulates doing a search on their home page for Web services. The format of the URL is: http://www.strikeiron.com/search?xyz, where xyz is the domain term to search for. The HTML results page is retrieved and parsed to pull out WSDL document locations.

The final menu selections in our toolkit are WSDL parsing and dynamic invocation selections, and WSIL parsing options. JWSDL is used to parse WSDL documents and WSIL4J to parse WSIL documents. Once a WSDL is picked, an Analyze button needs to be pressed. The service and inputs are displayed, and users can fill in parameters and invoke the web service

3. Experimental Results and Analysis

The experiments were conducted with a test base of 80 WSDL documents from unique domains. The original 40 WSDL documents were used to test and refine our algorithms. The test base includes WSDL documents found by: 1) Google searching; 2) other search engines; 3) random selections from www.xmethods.com, www.webserviceoftheday.com, www.strikeiron.com, and 4) Web services discovered while browsing the web.

The obvious approach of picking a UDDI entry for a business at random and then searching to discover the WSDL document had little success, mainly because often UDDI entries are company advertisements, rather than references to existing Web services. The UDDI entries picked for this study have been checked to verify that an existing WSDL document is associated with the entry. From our early experiments we learned that use of terms with highest frequency and key terms, such as terms from <title> tags, result in little success for WSDL and WSIL searches.

The measurement of results is based on whether a specific WSDL document is found for an input domain. We present a summary of results in Table 1. Overall the program resulted in a 97.4% success rate in discovery. The list of WSDLs to discover was created and used in the research for over a period of about 1 ½ years. WSDL document number 61, originally in the UDDI registry, can no longer be found there. Documents number 12 and 13 also no longer exist where originally found. Also, two WSDL documents (#46 and #49), while still found using the techniques described, do not provide the service anymore. They were both at one time registered in the StrikeIron directory. The total discovery results for the 77 cases remaining, after #12, 13, and 61 are removed, is 75 out of 77 successful discoveries, which yields approximately a 97.4% success rate of discovery. Table 2 shows the results for each individual method. The most successful method for this particular test pool is use of the StrikeIron directory search. Searching Google gives the next best rate of success. Combining StrikeIron and Google gives a success rate of 82.50% (66 of 80 found). UDDI searches, in some cases, found entries not located by other methods, and two WSDL documents could only be found if the domain name is tokenized (manually) for a UDDI category type search.

	Found	Not found	Percent found
First 40 cases	**36/40**	**4**	**90.0%**
First 38 (#12,13 removed)	36/38	2	95.0%
Reserved 40 cases	39/40	1	97.5%
Reserved 39 cases (#61 gone)	39/39	0	100.0%
Total for 77 cases after removing #12, 13, 61	75/77	2	97.4%

Table 1. Summary of WSDL Discovery results.

Use of a commercial free registry, StrikeIron, greatly increases the discovery rate. StrikeIron searching finds 11 WSDL documents that are not found by any other method (or 13.75%). Of the existing 79 WSDL documents, 2 cannot be found, even though they still exist. Entry #23 does not use the typical file name for a WSDL document. The Google-based techniques are expected to retrieve this document. However, it fails as it cannot find a name pattern match. Since the Google searches did not find it, this WSDL may not be linked into any web page or advertised, and may not be for general public use. Entry #36 was maintained by a doctoral student without any links to the WSDL page.

The success rate of our algorithms is high, especially considering that the test Microsoft UDDI registry was used for UDDI searches. Use of Google-based searches, through feeding queries to the Google web service, along with use of a popular privately maintained registry, supplemented by the UBR at Microsoft, provides WSDL documents for a high number of domains. Once UDDI is more widely accepted, as well as enhanced with semantics, Web service searches may become easier. Methods to enhance UDDI as well as WSDL documents with semantics have already been proposed. These will spur further research into intelligent dynamic invocation, allowing for better automation and interpretation of Web services.

4. Conclusions

The experiments in this study provide a glimpse into difficulties of finding Web services. Five methods are needed to obtain a 97.5% success rate, with the UDDI registry standard name search proving to be successful less than 13% of the time. As demonstrated in these experiments with the low UDDI success rates, use of the UDDI registry does not provide meaningful service information. This is expected to improve once semantic information is incorporated with the registry entries.

	Found	Found only by this method	% found out of 80	% found out of 77
Google method1 (inurl:)	**47**	**1**	**58.75%**	**61.04%**
Google method2 (site:)	27	1	33.75%	35.06%
UDDI by name	10	1	12.50%	12.99%
UDDI category search	28	4	35.00%	36.37%
UDDI category (tokenized)	30	2	37.50%	38.96%
UDDI total	34	1	42.50%	44.16%
StrikeIron directory	57	11	71.25%	74.03%
WSIL document	1	0	1.25%	1.30%
StrikeIron & Google	66		82.50%	85.71%

Table 2. Breakdown of Results by Method.

5. References

[1] Brittenham, Peter, "An Overview of the Web Services Inspection Language", IBM Developerworks, 2001.

[2] Skonnard, Aaron., "Publishing and Discovering Web Services with Disco and UDDI", MSDN Magazine, Feb 2002.

[3] Wang, Jiying, "Information Discovery, Extraction and Integration for the Hidden Web", Proc. VLDB PhD Workshop, 2003.

[4] Microsoft Web Services Dynamic Discovery (WS-Discovery) Specification, April 2005.

[5] IBM, Microsoft, Web Services Inspection Language (WS-Inspection 1.0), Nov. 2002