Clustering WSDL Documents to Bootstrap the Discovery of Web Services

Khalid Elgazzar, Ahmed E. Hassan, Patrick Martin
School of Computing, Queen's University, Canada
{elgazzar, ahmed, martin}@cs.queensu.ca

***Abstract*—The increasing use of the Web for everyday tasks is making Web services an essential part of the Internet customer's daily life. Users query the Internet for a required Web service and get back a set of Web services that may or may not satisfy their request. To get the most relevant Web services that fulfill the user's request, the user has to construct the request using the keywords that best describe the user's objective and match correctly with the Web Service name or location. Clustering Web services based on function similarities would greatly boost the ability of Web services search engines to retrieve the most relevant Web services. This paper proposes a novel technique to mine Web Service Description Language (WSDL) documents and cluster them into functionally similar Web service groups. The application of our approach to real Web services description files has shown good performance for clustering Web services based on function similarity, as a predecessor step to retrieving the relevant Web services for a user request by search engines.**

***Index Terms*—clustering WSDL documents, Web service, feature extraction, Web service clustering.**

I. Introduction

Service-oriented Architecture (SOA) has become a driving force for Web applications development. Service-oriented Computing (SOC) is a computing paradigm that is driven by SOA. SOC uses services as the basic constructs to support rapid, low-cost, and easy composition of distributed applications even in heterogeneous environments [1]. In SOA, a service is defined by a Web interface that supports interpretable operations between different software applications using a standard messaging protocol [2]. Web services are a popular implementation of SOA. A Web service is described by means of the Web Services Description Language (WSDL) [3], and that description is published in a public Universal Description Discovery and Integration (UUDI) registry. XML is used to construct the basic blocks of Web service communication by means of some form of XML messaging, such as Simple Object Access Protocol (SOAP) request or response or XML-Remote Procedure Call (XML-RPC).

Major providers of Web services, such as Google, Amazon and Yahoo, have decided to publish their Web services through their own websites instead of using public registries or brokers. This trend is forcing users to discover Web services using a search-engine model. Al-Masri et al. [4] show that services registered in public registries are decreasing in contrast with services crawled by search engines's crawlers. The authors point out that more that 53% of the UDDI Business Registry (UBR) registered services are invalid, whereas 92% of Web services cached by search engines are valid and active.

Searching for Web services using search engines, however, can result in a bottleneck in the discovery process, especially for non-semantic Web services because search engines do not understand the Web service functionalities outlined in the description file. Search engines partially match the search terms entered by the user with the Web service name, location, business, or tModel [5] defined in the Web service description file to get the results back. The use of these kinds of keywords are, by design, limited in WDSL specifications. If the search query does not contain part of the Web service name exactly, then the service may not be retrieved. It is therefore essential for the user to be aware of the concise and correct keywords in order to retrieve the most relevant services that match the request. This is difficult for users who are ultimately concerned with service functionality. A user may even miss services that use synonyms or variations of these keywords. For example, a service that contains *"car"* in its name may not be retrieved from a query looking for *"vehicle"*.

The problem of poor recall for search for non-semantic Web services using search engines can be approached in two ways. The first approach is to perform a broad matching process and return a potentially large number of unranked services, most of which may not be of interest to the user. The second approach is to improve search engine retrieval with mechanisms to cluster services into similar functional groups while they crawl their description files. This latter approach can effectively reduce the search space of Web services while improving the matching process. In addition, it can take advantage of the continuous crawling feature of search engines's crawlers by enabling adaptive re-clustering and self-organization to cope with the highly dynamic nature of Web services [6].

In this paper, we seek to improve Web services discovery with search engines by proposing a novel approach to clustering Web service description files (WSDL documents) into functionally similar groups prior to answering discovery requests.

Our main contributions are as follows:

- We present an approach that uses five key features extracted from WSDL documents in order to group Web services into functionality-based clusters.
- We experimentally demonstrate that our proposed approach outperforms (higher precision and recall) other approaches.

The rest of this paper is organized as follows. Section II gives a brief background on related work. Section III describes

the WSDL documents structure. Section IV introduces our proposed clustering approach. Section V explains how to extract features from WSDL documents. Section VI presents the integration of the extracted features to establish the relation between Web services. Section VII discusses the experiments and results. Finally, Section VIII concludes the paper and outlines future research avenues.

II. Background and Related Work

Research in Web mining has recently gained much attention due to the popularity of Web services and the potential benefits that can be achieved from mining Web services description files. Non-semantic Web services are described by WSDL documents while semantic Web services use Web ontology languages (OWL-S) [7] or Web Service Modeling Ontology (WSMO) [8] as a description language. Non-semantic Web services are more popular and supported by both the industry and development tools. The discovery process is quite different according to the Web services description method. Semantic Web services are discovered by high level match-making approaches [9], whereas non-semantic Web services discovery uses information retrieval techniques [10]. In our approach, we target the discovery of non-semantic Web services.

Nayak [5] proposes a method to improve the Web service discovery process using the Jaccard coefficient to calculate the similarity between Web services. He provides the user with related search terms based on other users' experiences with similar queries. In contrast, our approach clusters Web services based on their functionality in order to reduce the search space and improve query matching. We make use of five features extracted from the description files to calculate the similarity among Web services.

Many efforts have been made to overcome the drawbacks of UDDI-based discovery techniques. Guan et al. [11] propose two discovery mechanisms based on the cooperation between UDDI and Directory Facilitator (DF) to improve the efficiency of the UDDI-based Web service discovery. However, the study do not give enough evidence to show the effectiveness of their approach in improving the Web services discovery. Shuiguang et al. [12] propose an information model for registered services to improve the match-making in the discovery process. The authors use precision and recall to measure the performance of their approach. Xin et al. [13] propose the Web services search engine Woogle that is capable of providing Web services similarity search. Their engine, however, does not adequately consider data types, which usually reveal important information about the functionalities of Web services [14].

Liu and Wong [15] use a proposal similar to ours and apply text mining techniques to extract features such as service *content*, *context*, *host name*, and *name*, from Web service description files in order to cluster Web services. They propose an integrated feature mining and clustering approach for Web services as a predecessor to discovery, hoping to help in building a search engine to crawl and cluster non-semantic Web services. We differ in our choice of features. We believe that the *service context* and *service host name* features offer little help in the clustering process. Providers tend to advertise the services they provide on their own website, which means they provide different Web services on the same site. Hence, mining the surrounding Web pages (*service context*) or considering the *host name* does not help with the meaning of the Web service, which is not the case in UDDI. In addition, some Web services do not make use of the `<documentation>` element in the WSDL document, which means there is insufficient information for the *content* feature. Relying on attributes' names as a Web service content may also be misleading since the names may not follow any naming conventions and they may not be descriptive or even correct English words.

III. WSDL Document Structure

WSDL is an XML-based language that provides a model for describing non-semantic Web services [3]. It is also used to describe the operations that can be performed by a certain Web service as well as its location. In our feature extraction we consider the structure of WSDL specification version 1.1 as it is most often supported by software development tools.

A WSDL document describes a Web service using six major components:

- `<types>` element is an XML type definition that describes the data containers used in message exchanges.
- `<messages>` element is an abstract representation of the transmitted information. Typically, a message contains one or more logical parts (parameters). These parts are associated with a type definition.
- `<portType>` is an important component in WSDL documents, in which a set of abstract operations (functions) that can be performed by the Web service are defined. Each operation is associated with an input and/or output message.
- `<binding>` component specifies the communication protocol and data format for each operation and message defined in a particular portType element.
- `<service>` element is a composite operation that aggregates multiple related ports or functions.

IV. Proposed Clustering Approach

Our proposed approach is inspired by the information available in Web services description document. We mine the WSDL documents to extract features that describe the semantic and behavior of the Web service, specifically the *WSDL content*, *WSDL types*, *WSDL messages*, *WSDL ports* and the *Web service name*. These features describe and reveal the functionality of a Web service [13]. Integrating these features together, we cluster Web services into functionally similar groups. This is a predecessor step to assist a service search engine in identifying the Web service functionality and match Web services with users' requests. Figure 1 illustrates graphically our proposed approach to cluster WSDL documents to aid a service search engine. A search engine's crawler crawls WSDL documents from the internet and applies offline our proposed clustering approach to group similar functionally services. When a user queries the service search engine for a

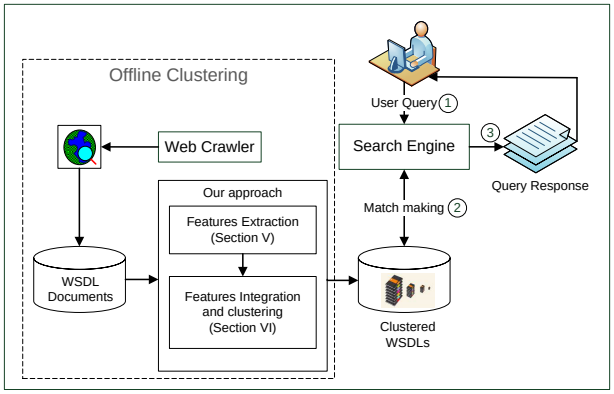

Fig. 1. Schematic block diagram illustrates the big picture that inspired our WSDL clustering approach.

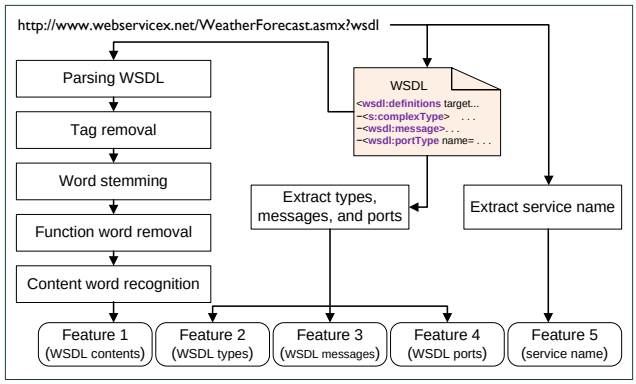

Fig. 2. Block diagram of the features extraction process.

desirable objective (step 1), it uses the clustered Web services to match semantically the query (step 2) and return the most relevant Web services (step 3) that satisfies the requested objective.

V. Extracting Features From WSDL Documents

In this Section we describe how we extract the five proposed features from WSDL documents. Figure 2 illustrates the steps for feature extraction.

Feature 1: WSDL Content

We begin by reading the WSDL document contents directly from the WSDL URI. Each WSDL document f_i describes a Web service s_i. We process the contents of the WSDL document in order to extract a vector of meaningful content words for the Web service s_i. Our approach to building the vector consists of the following five steps (shown in Figure 2):

1) **Parsing WSDL:** The contents of the document are parsed based on white spaces to produce a vector of tokens T_i.
2) **Tag removal:** The next step removes all tokens from T_i that are part of a XML tag so that only valid content words remain in the vector. Removing XML tags from the tokenized vector is straightforward since all XML tags used in a WSDL document are predefined.
3) **Word stemming:** In this step, all words in T_i are reduced to their base words using a Porter stemmer [16]. Tokens with a common stem will usually have the same meaning, for example, 'connect', 'connected', 'connecting', 'connection', and 'connections' all have the same stem 'connect'. Having one or all of them will not make a difference in terms of word variations in the semantic of a Web service. However, words which appear more often are more important than others. We consider the number of occurrence in the following steps.
4) **Function word removal:** Function words tend to be independent of one another. Often, function words can be distinguished from content words using a Poisson distribution to model word occurrence in documents[15]. This step is intended to remove all function words from the service word vector. To decide whether a certain word w is a function word, we calculate the overestimation factor for all words in the word vector as follows:

$$\Lambda_w = \frac{estimatedDocumentFreq}{observedDocumentFreq} = \frac{\hat{n}_w}{n_w} \qquad (1)$$

where n_w is the number of documents that contain the word w (we use the occurrence in Web documents), $\hat{n}_w$ and is the number of estimated documents that contain the word w. To estimate the document frequency for a word we need to have the document frequency for a single occurrence of that word in the Web corpus. While it is not feasible for a search engine to identify documents that contain a single occurrence of a particular search term, there are techniques, such as the K-mixture word distribution model [17], to estimate the word frequency using page count. In this paper, we use the Yahoo search engine to obtain a single occurrence page count for a certain search term as follows. We first search the Web using the Yahoo search engine for all pages that contain the desired word, for example *"weather"*. This gives the Web document frequency regardless of how many times the search term *"weather"* appears in a document. We then search using the term *"weather * weather"*, which gives all documents containing at least two occurrences of the word *"weather"*. The difference between these two page counts is an estimate for the Web document frequency for a single occurrence of the term *"weather"*. We can then calculate the overestimation factor (discussed in more detail in [15]) for all words in T_i as well as the average $avg[\Lambda]$ of all overestimation factors. An overestimation factor threshold (Λ^{thre}) is defined as follows [15].

$$\Lambda^{thre} = \begin{cases} avg[\Lambda] & \text{if } avg[\Lambda] > 1 \\ 1 & \text{otherwise} \end{cases} \qquad (2)$$

Any word that has an overestimation factor above the Λ^{thre} is considered to be a content word. Otherwise the

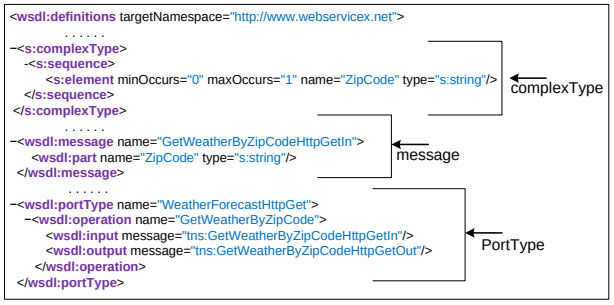

Fig. 3. An excerpt from the WeatherForecast Web service (http://www.webservicex.net/WeatherForecast.asmx?wsdl) that shows the structure of types, messages, and ports.

word is considered to be a function word and is removed from the vector T_i.

5) **Content word recognition:** WSDL documents usually contain general computing content words such as *'data'*,*'web'*,*'port'*, etc. These words appear in most Web service description files and so can not be used to discriminate between Web services. The objective of this step is to remove words that do not describe the specific semantics of the Web service. We first apply the k-means clustering algorithm [18] with $k = 2$ on T_i to cluster the remaining words into two groups, one group describing the meaning of the Web service and the other group is for general computing words. We use K-means because it is simple, fast, and efficient if the number of clusters is known beforehand. We use Normalized Google Distance (NGD) [19] as a featureless distance measure between words. We use a cluster selection method to automatically recognize which cluster contains the Web-service-specific words. A number of sophisticated cluster selection algorithms have been proposed such as the one based on iterative propagation of penalty weights [15]. We chose, however, to use a simple approach based on calculating the average NGD between each of the two clusters and a predefined vector of general computing words such as $\{$ *runtime, bind, web, service, module, data, post, developer* $\}$. The cluster closest to this oracle is determined to be the non-Web-service-specific cluster and its words are removed from the word vector T_i.

Feature 2: WSDL Types (complexType)

WSDL documents contain a section that defines data containers which will be used by messages to transmit information between Web services. Figure 3 shows an example of a complexType definition that appears in the WeatherForecast Web service. WSDL specifications use XML Schema Definition (XSD) as their canonical type system. Types can be as simple as a single element or as complex as an array of elements. Each element has a name attribute and a type attribute. While the name attribute is sometimes not a useful feature, the type attribute is a good candidate for describing the functionality of a service. Xin et. al [13] show that complex data types are the most informative element in WSDL documents. We therefore extract element types and determine the number of type matches between a pair of Web services using

$$match(s_i, s_j) = \frac{M(s_i, s_j)}{avg(E_{s_i}, E_{s_j})} \quad (3)$$

where s_i and s_j are different Web services, $M(s_i, s_j)$ is the number of matched types between Web services s_i, s_j, and E_{s_i} and E_{s_j} are the total numbers of defined types in Web service s_i and s_j respectively.

Feature 3: WSDL Messages

Messages encompass one or more logical parameters. Each parameter is associated with one of the system types. Multiple parameters (part elements) are used if the message has multiple logical units, such as a message containing a purchase order with order items and its invoice. Message definitions are typically considered as an abstract definition of the message content. A message binding section defines how the abstract content is mapped into a concrete format [3]. Figure 3 shows an example of a simple message definition that appears in the WeatherForecast Web service. The message may contain multiple parts and the order in which these parts appear is important to the message definition. In our approach, we match the messages' structure between Web services and use Equation (3) to calculate this match. In this case, $M(s_i, s_j)$ is number of matched messages between Web services s_i and s_j, and E_{s_i}, E_{s_j} are the total number of defined messages in Web services s_i, and s_j respectively.

Feature 4: WSDL Ports

A `<portType>` defines the combination and sequence of messages for an operation. WSDL 1.1 supports four types of message exchange patterns [20]:

- One-way: The service receives a single input message.
- Request-response: The service receives a request message and responds with an output message.
- Solicit-response: The service first sends an output message and then waits for an input message in response.
- Notification: The service sends an output message without waiting for anything in return as, for example, in the case of state updates.

Figure 3 illustrates a `<portType>` definition section in the WeatherForecast Web service. We evaluate how many portTypes are similar, with respect to both message sequence and message structure, between two Web services using Equation (3).

Feature 5: Web Service Name

We consider the Web service name used in the URI of the WSDL document. For example, the URI of the WeatherForecast Web service is `http://www.webservicex.net/WeatherForecast.Asmx?WSDL`, and so the name of the Web service is "Weather Forecast". This name could be totally different from the name used inside the WSDL document itself. In

case of composite names, such as 'WeatherForecast', we split the composite name into multiple names based on the assumption that a capital letter indicates the start of a new word. Using NGD we find the similarity between services names as follows:

$$sim(sname_i, sname_j) = 1 - NGD(sname_i, sname_j) \quad (4)$$

where $sname_i$ and $sname_j$ are the names of the Web services s_i and s_j respectively.

VI. Feature Integration

We use the Quality Threshold (QT) clustering algorithm [21] to cluster similar Web services based on the five similarity features presented above. We decided to use QT because it returns consistent results across multiple runs with the same input, and it can be used to cluster particular groups. The drawback of QT, however, is that it is more computationally expensive than other clustering algorithms [21]. We measure the similarity factor $\Theta(s_i, s_j)$ between Web services s_i and s_j as follows:

$$\begin{aligned}\Theta(s_i, s_j) =& 0.2S(T_i, T_j) + 0.2sim(sname_i, sname_j) + \\ & 0.2match(typ_i, typ_j) + 0.2match(msg_i, msg_j) + \\ & 0.2match(port_i, port_j) \end{aligned} \quad (5)$$

$\Theta(s_i, s_j)$ is equal to "1" if the two services are identical and $\Theta(s_i, s_j)$ is equal to "0" if they are completely different. We normalize $\Theta(s_i, s_j)$ by assigning weights of 0.2 to each of the five similarity features. We determined experimentally that these weights give reasonable results. In Equation (5) T_i and T_j are the content word vectors of services s_i, s_j respectively. $S(T_i, T_j)$ is the average similarity between the content word vectors T_i, and T_j and is calculated with

$$S(T_i, T_j) = \frac{\sum_{a \in T_i} \sum_{b \in T_j} sim(a, b)}{|T_i||T_j|} \quad (6)$$

where $sim(a, b)$ is the featureless similarity factor computed between words a and b using NGD based on the word co-existence in Web pages. $sim(a, b)$ is calculated using

$$sim(a, b) = 1 - NGD(a, b) \quad (7)$$

where a and b are the content vector words belong to T_i and T_j respectively.

VII. Experiments and Results

We use two criteria to evaluate the performance of our approach, namely Precision and Recall. "Precision can be seen as a measure of exactness or fidelity, whereas Recall is a measure of completeness" [22]. Precision and Recall have been often used to evaluate information retrieval schemes [12]. We extend the use of these two measures to evaluate our approach as follows:

$$precision = \frac{\sum_{i \in C} P_{c_i}}{length(C)}, P_{c_i} = \frac{succ(c_i)}{succ(c_i) + mispl(c_i)} \quad (8)$$

$$recall = \frac{\sum_{i \in C} R_{c_i}}{length(C)}, R_{c_i} = \frac{succ(c_i)}{succ(c_i) + missed(c_i)} \quad (9)$$

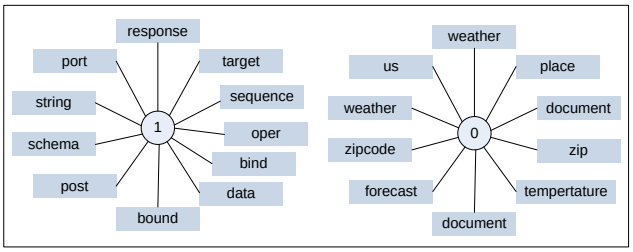

Fig. 4. The output of the content words recognition phase for the WeatherForecast Web service.

where c_i is the cluster i, P_{c_i} and R_{c_i} are precision and recall for cluster c_i respectively, $succ(c_i)$ is the number of Web services successfully placed in the proper cluster c_i, $mispl(c_i)$ is the number of Web services that are incorrectly clustered into c_i (i.e. they should not be there), $missed(c_i)$ is the number of Web services that should be clustered into c_i but are incorrectly placed in other clusters, and $length(C)$ is the number of clusters.

Our experiments are based on the WSDL documents of 400 online Web services gathered from real-world Web service providers and Web service repositories such as WebserviceList, WebserviceX, and xMethods. We do not download each WDSL document, instead, our approach reads the contents directly from the WSDL document URI. We perform a manual classification of the WSDL documents to serve as a comparison point for the clustering algorithms. We distinguish the following categories: "Currency exchange", "Weather", "Address validation", "E-mail verification", and "Credit card services" as shown in Table I.

We first mine the contents of the WSDL documents to extract the content words vector T_i where $1 \geq i \geq 400$, describing the meaning of the Web services. Using the java library WVTool [1], each line is uploaded as an instance of *WVTDocumentInfo* class. The output is a vector of words without the XML tags. These vectors are taken to the next phase using the Porter stemmer to reduce all the words in each vector to their roots. Then we calculate the overestimation factor for all the words in each vector to distinguish between function words and content words. We use the Yahoo search engine to find the document frequency and estimated document frequency for each word. We next identify the content words for the Web services by clustering each word vector into two groups using the k-means clustering algorithm, in which we use NGD as a featureless similarity measure between words. Figure 4 shows a sample output of this phase for the WeatherForecast Web service. Finally, we use a cluster selection algorithm to pick out the cluster of non-content words by calculating the clusters' similarity, using Equation (6), with a group of general computing words including {*runtime, bind, web, service, module, data, post, developer*}. We create the *types* feature for the Web services by extracting all the defined *complexTypes* along with their

[1] `http://wvtool.sf.net`

TABLE I
THE MANUALLY IDENTIFIED CATEGORIES FOR CLUSTERING VERIFICATION.

Category	WSDL URI	
Currency exchange (19)	http://www.atlaz.net/webservices/GetCurrencyExchange.wsdl	http://www.webservicex.net/CurrencyConvertor.asmx?WSDL
	http://server1.pointwsp.net/ws/finance/currency.asmx?WSDL	http://allysoft.ru/BScurrency/currency.asmx?WSDL
	http://www.freewebs.com/jimmy_cheng/CurrencyExchangeService.wsdl	http://fx.cloanto.com/webservices/CurrencyServer.asmx?wsdl
	http://www.currencyserver.de/webservice/currencyserverwebservice.asmx?WSDL	http://currencyconverter.kowabunga.net/converter.asmx?WSDL
	http://ws.soatrader.com/gama-system.com/1.0/CurrencyExchangeRates?wsdl	http://tvazteca.viajez.com/WServicesDev/CurrencyRequest?WSDL
	http://ws.serviceobjects.com/ce/CurrencyExchange.asmx?WSDL	http://ws2.serviceobjects.net/ce/CurrencyExchange.asmx?WSDL
	http://www.petermeinl.de/CurrencyConverter/CurrencyConverter.asmx?wsdl	http://currency.niekutis.net/currency.asmx?wsdl
	http://cs.daenet.de/webservice/CurrencyServerWebService.asmx?WSDL	http://www.xignite.com/xCurrencies.asmx?wsdl
	http://trial.serviceobjects.com/ce/CurrencyExchange.asmx?WSDL	http://ws.strikeiron.com/ForeignExchangeRate3?WSDL
	http://ws.soatrader.com/baydonhill.com/0.1/Currency?wsdl	
Weather (16)	http://www.webservicex.net/globalweather.asmx?wsdl	http://www.webservicex.net/usweather.asmx?wsdl
	http://www.webservicex.net/WeatherForecast.asmx?wsdl	http://www.deeptraining.com/webservices/weather.asmx?WSDL
	http://ws.soatrader.com/wopos.com/0.1/Weather?wsdl	http://ws365.net/ws/weather.asmx?WSDL
	http://ws.soatrader.com/cs.uga.edu/0.1/WeatherFetcher?wsdl	http://weather.shellware.com/weather.asmx?WSDL
	http://asyncpostback.com/WeatherService.asmx?WSDL	http://ws.soatrader.com/bea.com/0.1/weather?wsdl
	http://weather.cobbnz.com/weatherservice/webservice.asmx?wsdl	http://209.162.186.60/globalweather.asmx?WSDL
	http://trial.serviceobjects.com/fw/FastWeather.asmx?WSDL	http://api.wxbug.net/weatherservice.asmx?wsdl
	http://www.weather.gov/forecasts/xml/DWMLgen/wsdl/ndfdXML.wsdl	http://lostsprings.com/weather/WeatherService.asmx?WSDL
Address validation (16)	http://services.postcodeanywhere.co.uk/us/lookup.asmx?wsdl	http://ws.strikeiron.com/GlobalAddressLocator3?WSDL
	http://ws.fraudlabs.com/postalcodeworldMexico_webservice.asmx?wsdl	http://ws.strikeiron.com/GlobalAddressVerification4?WSDL
	http://ws.fraudlabs.com/zipcodeworldUS_webservice.asmx?wsdl	http://ws.serviceobjects.com/av/AddressValidate.asmx?WSDL
	http://validator2.addressdoctor.com/addInteractive/Interactive.asmx?WSDL	http://ws.cdyne.com/psaddress/addresslookup.asmx?wsdl
	http://trial.serviceobjects.com/avca/ValidateCanada.asmx?WSDL	http://arcweb.esri.com/services/v2/AddressFinder.wsdl
	http://ws.fraudlabs.com/postalcodeworldCanada_webservice.asmx?wsdl	http://ws.strikeiron.com/ZIPPostalCodeInfo5?WSDL
	http://ws.soatrader.com/welho.fi/0.1/AddressAutoCompleteService?wsdl	http://ws.soatrader.com/servicex.co.uk/0.1/Address?wsdl
	http://142.176.62.103/GEONOVA_WS/CivicAddressPointRange.asmx?WSDL	http://ws.fraudlabs.com/areacodeworldwebservice.asmx?wsdl
E-mail verification (8)	http://ws.xwebservices.com/XWebEmailValidation/XWebEmailValidation.asmx?wsdl	http://ws.cdyne.com/emailverifyws/emailverify.asmx?wsdl
	http://www.siprod.net/webservices/xemail/xemailwebservice.asmx?WSDL	http://trial.serviceobjects.com/ev2/emailvalidation2.asmx?WSDL
	http://ws2.fraudlabs.com/mailboxvalidator.asmx?wsdl	http://soap.towerdata.com/validate.wsdl
	http://ws.cdyne.com/emailverify/Emailvernotestemail.asmx?wsdl	http://ws.strikeiron.com/EmailVerify?WSDL
Credit card check (10)	http://webservices.tiscali.com/CreditCardServices.asmx?wsdl	http://www.webservicex.net/CreditCard.asmx?wsdl
	https://webservices.optimalpayments.com/creditcardWS/CreditCardService/v1?wsdl	http://www.webservicex.net/CreditCard.asmx?wsdl
	http://www.cbr.ru/CreditInfoWebServ/CreditOrgInfo.asmx?wsdl	http://webservices.primerchants.com/creditcard.asmx?wsdl
	http://secure.cdyne.com/creditcardverify/luhnchecker.asmx?wsdl	http://webservices.primerchants.com/creditcard.asmx?WSDL
	http://ws.strikeiron.com/FraudLabs/CreditCardFraudDetection?WSDL	http://ws.fraudlabs.com/fraudlabswebservice.asmx?wsdl

TABLE II
CALCULATING COMPLEXTYPE MATCHES BETWEEN FOUR WEB SERVICES.

Service	complexType			complexType matches			
	element	name	type	WS1	WS2	WS3	WS4
WS1	GetWeather	CityName	string	1.0	0.86	0.67	0.25
		CountryName	string				
	GetWeatherResponse	GetWeatherResult	string				
	GetCitiesByCountry	CountryName	string				
	GetCitiesByCountryResponse	GetCitiesByCountryResult	string				
WS2	GetWeatherReport	ZipCode	string		1.0	0.80	0.26
	GetWeatherReportResponse	GetWeatherReportResult	string				
	null	string	string				
WS3	GetWeather	City	string			1.0	0.29
	GetWeatherResponse	GetWeatherResult	string				
WS4	testAddressResponse	testAddressReturn	string				1.0
	displayAddress	x	double				
		y	double				
		scale	double				
		objectId	long				
		imgHeight	int				
		imgWidth	int				
		dpi	long				
	displayAddressResponse	displayAddressReturn	string				
	testAddress	objectId	long				
		centreX	double				
		centreY	double				

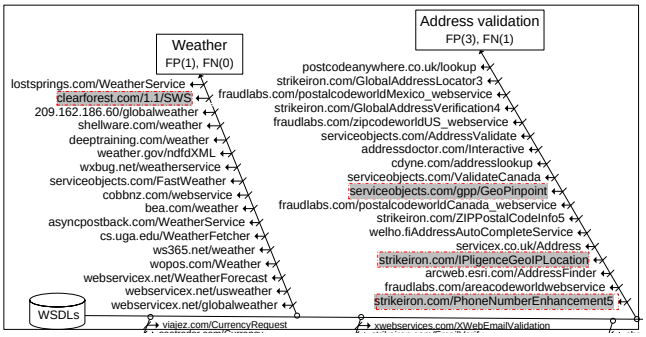

Fig. 5. A snippet from the our clustering approach output focuses on the identified groups and indicating *false positives*(FP) and *false negatives*(FN).

elements in each WSDL document and determine the number of matching elements for every pair of Web services using Equation (3). Table II shows the results for four sample Web services, `webservicex.net/globalweather` (WS1), `webservicex.net/usweather` (WS2), `deeptraining.com/weather` (WS3), and `nsw.gov.au/AddressImageWS` (WS4). We also extract the information for the *messages* and *ports* features and calculate the similarity for each pair of services.

We cluster the feature sets of the 400 Web services using QT clustering algorithm, which uses the composite relatedness measure in Equation (5) to calculate similarity between Web services. We use 0.7, which was determined experimentally, as the minimum similarity threshold between any pair of services in a cluster. Figure 5 shows a snippet from the clustering output, which focuses on the groups that we identified manually. In each cluster there may be some Web services incorrectly placed in it (*false positives*, which are marked by a dark background in Figure 5), as well as others that are supposed to be there but are placed in other clusters (*false negatives*). For example, Web services described by `http://ws.strikeiron.com/IPligenceGeoIPLocation?WSDL`,

http://trial.serviceobjects.com/gpp/GeoPinpoint.asmx?WSDL, and http://ws.strikeiron.com/PhoneNumberEnhancement5?WSDL are incorrectly placed in the "Address validation" group, whereas, the Web service from the same group that is described by the WSDL document http://142.176.62.103/GEONOVA_WS/CivicAddressPointRange.asmx?WSDL is misplaced in another cluster. Examining the WSDL documents of these Web services we see that the first three Web services are either mapping *ip addresses* to geographic locations or enhancing phone directories by associating addresses to phone records. The descriptive text inside the WSDL documents leads to a confusion with civic addresses and the WSDL contains the same complexTypes and elements structures of the "Address validation" group. That is why our approach added them incorrectly to this group. On the other hand, our approach incorrectly clusters the service with the WSDL document http://142.176.62.103/GEONOVA_WS/CivicAddressPointRange.asmx?WSDL because the document defines addresses in a different manner than the other Web services and it uses a large number of acronyms.

For the sake of performance comparison, we implemented the clustering approach proposed by Liu and Wong [15] and applied it on our dataset. This proposed clustering mechanism relies on less information from the description files and mines for features from surrounding pages of the Web service and the host name. We used the Quality Threshold clustering algorithm in our implementation instead of the Tree-Traversing Ant (TTA) algorithm used in the original paper as QT is the one we used in our approach. The results shows that, for example, services described by http://www.webservicex.net/ConvertTemperature.asmx?wsdl, http://weather.terrapin.com/axis2/services/HurricaneService?wsdl, http://ws.soatrader.com/syromlya.ru/0.1/Prediction?wsdl, and http://webservices.daehosting.com/services/TemperatureConversions.wso?WSDL, are incorrectly placed the "Weather" cluster. Investigating these WSDL documents, we see that the first three do not make use of the <documentation> element to provide a description of the Web service's functionality and their attributes' names can be confused with the weather Web services. The fourth of the above incorrectly placed Web services has text to describe its functionality but uses weather-like terms in the description. In all these services, however, the <complexType> definitions and <message> structures are completely different from the other weather Web services, which explains why our approach is able to recognize them.

On the other hand, the Web service described by http://ws.fraudlabs.com/fraudlabswebservice.asmx?wsdl, for example, is not correctly placed in the "Credit card services" cluster by either approach. A closer look to this Web service shows that it does not have a <documentation> element and neither its name nor its contents imply that it provides a credit check. It is worthwhile noting that our approach failed to correctly cluster this service because it uses very different <complexType> and <message> structures than the ones used by other credit card Web services.

TABLE III
PERFORMANCE MEASURES RELATED TO THE FIVE IDENTIFIED CLUSTERS.

Cluster	Our approach		[15]'s approach	
	Precision%	Recall%	Precision%	Recall%
Currency exchange	90.0	94.7	84.2	88.9
Weather	94.1	100	70.0	87.5
Address validation	83.3	93.7	60.0	93.7
E-mail verification	80.0	100	58.3	87.5
Credit card services	90.0	90.0	60.0	90.0

TABLE IV
EVALUATING THE SIGNIFICANCE OF DIFFERENT CLASSIFICATION FEATURES.

Cluster	Precision%	Recall%
Currency exchange	82.6	78.9
Weather	72.3	93.7
Address validation	60.0	93.3
E-mail verification	56.0	77.7
Credit card services	60.0	88.9

Table III shows the performance comparison (in terms of the precision and recall) for our approach versus Liu and Wong's approach [15] relative to the five manually identified groups in our test set of Web services. We could not calculate the overall precision and recall since we could not manually identify all clusters in our dataset. We note that the low precision of our approach for the "Address validation" and "E-mail verification" clusters in Table III is due to the mutual correlation between these two groups as well as some individual services from the *ip-to-country* Web services domain. We also note that all Web services that are supposed to belong to "E-mail verification" and "Weather" groups are successfully placed in the clusters, as indicated by 100% recall value.

Looking at the performance results of the two approaches in Table III, we note that our approach has higher precision and higher recall for all the identified categories. For example, our approach improved the precision for the "Address validation" group by 23.3% and 30% for "Credit card services" group.

We also conducted an experiment to evaluate the significance of the common features used by our approach and Liu and Wong's approach on the classification process. We ran our approach clustering only the *service name* and the *content* features. Table IV shows the performance measure form this experiment. Comparing the results in Table IV and Table III, we conclude that, performance is slightly increased, in general, by adding *context* and *host name* features. However, sometimes precision gets better and recall becomes higher if *context* and *host name* are not considered such as the case of increasing the precision from 70.0% to 72.3% for the "Weather" group. We believe that this negative contribution of these two features is due to the fact that Web services are published through providers' website or public access uncategorized repositories, in which the surrounding pages of the WSDL documents have no relations to the functionality of

the Web services that described by these WSDLs. On the other hand, our approach performs well by adding features such as *complexTypes*, *messages* and *ports*. The impact of these features improved the clustering reliability by increasing our precision and enhancing our recall.

VIII. Conclusion

Effective Web service discovery is an important issue, especially for non-semantic Web services. Traditional UDDI-based and search engine-based Web service discovery lacks the ability to recognize the content of the Web service description file. In this paper, we propose an approach to improve service discovery of non-semantic Web services by clustering similar services through mining WSDL documents. We identify five key features that are extracted and integrated in order to group Web services into functionality-based clusters.

Our clustering approach can be integrated into search engines to improve the quality of Web service discovery by helping to identify the Web services relevant to a user request. This will, in turn, add value to the discovery process by providing users with better quality options in selecting a service. Experiments show a performance improvement in the quality of the retrieval compared with previous approaches. As future work, we plan to improve features integration by choosing optimized weights for each feature using a linear programming approach.

References

[1] Michael P. Papazoglou, Paolo Traverso, Istituto Ricerca, Scientifica Tecnologica, " Service-Oriented Computing: State of the Art and Research Challenges," *Computer*, VOL. 40, NO. 11, pp 38-45, Nov. 2007.

[2] "Web Services Architecture," February 11, 2004. [online] Available: `http://www.w3.org/TR/ws-arch`. [Accessed: Feb. 26, 2010].

[3] "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language," June 26, 2007. [Online]. Available: `http://www.w3.org/TR/wsdl20`. [Accessed: Feb. 26, 2010].

[4] Eyhab Al-Masri, Qusay H. Mahmoud, "Investigating web services on the world wide web," *International World Wide Web Conference (WWW 2008)*, pp. 795-804, 2008.

[5] Richi Nayak, " Data mining in Web services discovery and monitoring ," *International Journal of Web Services Research*, Vol. 5, No. 1, pp. 63-81, January, 2008.

[6] Wei Liu, "Trustworthy Service Selection and Composition Reducing the Entropy of Service-oriented Web," *3rd International Conference on Industrial Informatics(INDIN 2005)*, pp. 104-109, 2005.

[7] M. Burstein, J. Hobbs, O. Lassila, D. Mcdermott, S. Mcilraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, K. Sycara, D. Martin (ed.) "OWL-S: Semantic Markup for Web Services," W3C Member Submission, 2004.

[8] H. Lausen, A. Polleres, "Web Service Modeling Ontology (WSMO)," W3C Member Submission, 2005.

[9] Matthias Klusch, Benedikt Fries, Katia Sycara, "Automated semantic web service discovery with owls-mx," *Proceedings of 5th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2006.

[10] John D. Garofalakis, Yannis Panagis, Evangelos Sakkopoulos and Athanasios K. Tsakalidis, "Contemporary Web Service Discovery Mechanisms," *Journal of Web Engineering*, Vol. 5, No. 3, pp. 265-290, September 2006.

[11] Guan Hong-Jie, Meng Fan-Rong, Sun Jin-Fei, Du Peijun, "Web service discovery based on the cooperation of UDDI and DF," *4th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM)*, pp. 1-4, 2008.

[12] Shuiguang Deng, Zhaohui Wu, Jian Wu, Ying Li, Jianwei Yin, "An Efficient Service Discovery Method and its Application," *International Journal of Web Services Research*, Vol. 6, No. 4, pp. 94-117, 2009.

[13] Xin Dong, Alon Halevy, Jayant Madhavan, Ema Nemes, Jun Zhang, "Similarity Search for Web Services," *Proceedings of the 30th VLDB Conference, Toronto, Canada*, pp. 372-383, 2004.

[14] Natallia Kokash, "A Comparison of Web Service Interface Similarity Measures," *Frontiers in Artificial Intelligence and Applications*, Vol. 142, pp.220-231, 2006.

[15] Wei Liu, Wilson Wong, "Web service clustering using text mining techniques," *International Journal of Agent-Oriented Software Engineering*, Vol. 3, No. 1, pp. 6-26, 2009.

[16] M. F. Porter, "An Algorithm for Suffix Stripping", *Program*, Vol. 14, No. 3, pp. 130-137, 1980.

[17] Slava M. Katz, "Distribution of content words and phrases in text and language modeling," *Natural Language Engineering*, Vol. 2, No. 1, pp. 15-59, March 1996.

[18] Jain AK, Dubes RC, *Algorithms for clustering data*. Prentice-Hall, Englewood Cliffs, 1988.

[19] Cilibrasi, Rudi L, Vitnyi, Paul M. B., " The Google similarity distance," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 19, No. 3, pp. 370-383, March 2007.

[20] Ethan Cerami, *Web Services Essentials*. O'Reilly & Associates, ISBN:0-596-00224-6, February 2002.

[21] Laurie J. Heyer, Semyon Kruglyak, Shibu Yooseph, "Exploring Expression Data: Identification and Analysis of Coexpressed Genes," *Genome Research*, Vol. 9, No. 11, pp. 1106-1115, November 1999.

[22] John Makhoul, Francis Kubala, Richard Schwartz, Ralph Weischedel, "Performance measures for information extraction," *DARPA Broadcast News Workshop, Herndon, VA*, February 1999.