Ontological Representation of Public Web Services

Maricela Bravo
Systems Department
UAM-Azcapotzalco
DF, Mexico
mcbc@correo.azc.uam.mx

Mónica Silva-López
Systems Department
UAM-Azcapotzalco
DF, Mexico
misl@correo.azc.uam.mx

Blanca Silva-López
Systems Department
UAM-Azcapotzalco
DF, Mexico
rbsl@correo.azc.uam.mx

***Abstract*—Among the main benefits of service-oriented architectures is the reutilization of software components that may solve specific tasks for complex problems, requiring the composition of multiple Web services. Currently Internet is largely populated with Web services offered by different providers and published in various Web repositories. However, public available Web services still suffer from problems that have been widely discussed, such as the lack of functional semantics. This lack of semantics makes very difficult the automatic discovery and invocation of public Web services, even when the system integrator can obtain a copy of the WSDL file. This paper describes an ontological approach for discovering similarity relations between public Web services. The objective of this work is to extract relevant data that is coded into service operations descriptions, calculate similarity measures between them, represent the discovered similarities in an ontological form, and execute inference. Experimental results show that the overall process towards the automation of public Web services discovery based on ontology population and structural similarity measures is feasible and can be completely automated.**

Keywords-Web services; Structural Similarity Measures; Similarity Relations Discovery; Automated Ontology Population; and Inference.

I. INTRODUCTION

In the last decade, many software vendors have developed, deployed and offered software as services using interface description languages, such as the Web Service Description Language (WSDL) [1]. In order to make their services available online, providers publish their service descriptions in public Web service repositories, which may or may not be conformant to a specific standard such as UDDI [2] or ebXML [3]. When software integrators search for Web services that meet certain criteria in public repositories, and try to select and invoke existing Web services, they may face some of the following problems:

Lack of well-documented Web service descriptions. This is a common problem that many public Web service clients or requestors face. A study and report of this problem was presented by Rodríguez et al. [4]. In this work, authors identify common mistakes in WSDL documents: inappropriate or lacking comments, use of ambiguous names for the main elements, redundant port-types, low cohesive operations in the same port-type, enclosed data model, redundant data models, etc. According to [4], less than 50% of the studied WSDL files have some documentation. Additionally, the naming of services, operations, messages and parameters does not follow any convention, and there is no obligation to provide additional semantic information. These reasons cause enormous difficulties during search, selection and invocation of services.

Lack of semantically enhanced Web services repositories. There are many public Web service repositories, but they do not offer sufficient semantic information about the service functionality, making very difficult the automated exploitation of deployed Web services. Majority of public repositories offer key-based search mechanisms, and some sort of classifications, but none of them offer correlations discovery between existing services considering provided interface (template) information.

Different solutions have been proposed to solve these limitations. The semantic Web has influenced many works by providing logic-based mechanisms to describe, annotate and discover Web services. Within this context, McIlarith et al. [5] proposed one of the first initiatives to markup Web services based on DAML (ontology language), which started the important research area of "Semantic Web Services". The term Semantic Web Services is related to the set of technologies and models based on the implementation and exploitation of ontologies as a mechanism to semantically enhance service descriptions, for instance: OWL-S [6], WSMO [7], and SAWSDL [8]. However, these methods require human experts intervention to construct ontologies and annotate Web service descriptions before their deployment.

From the perspective of Web services providers, if they want to take advantage of these semantic-based technologies, they will have to re-design their solutions with the following considerations: in case of annotating semantically their Web services using SAWSDL, they need to construct or select an ontological representation relative to the domain of the services offered; in the case of using OWL-S, service providers need to learn this model and use the tools available to create the corresponding ontological descriptions of their services; and in case of using WSMO, the learning curve is steep because it requires more effort to understand and use the complex framework of WSMO with a new ontology language. Considering that service providers are familiar with software development, but not necessarily with ontologies or semantic Web technologies.

 ISBN: 978-1-61208-266-0

From the point of view of a service requestor, the first problem that he will face is to find Web services repositories containing semantic Web service descriptions using these technologies. Furthermore, he will not find a universal repository containing all types of semantic service descriptions. Until now there is no reported approach or tool that automatically translates or connects (without human intervention) any pre-existing Web service description to any of the aforementioned solutions.

Despite the increasing popularity of semantic-based technologies, the numerous researchers devoted to them, the great advances and achievements; there is still an important gap between these semantic-based Web service technologies and the pre-existing Web services, which were deployed using only WSDL (including the common mistakes pointed by Crasso). There is no doubt that the semantic Web trend will continue and will consolidate in the following years, and if service providers want to stay competitive, they need to adapt and re-deploy their services using these technologies. Meanwhile, from the perspective of computer science, many solutions can be designed and constructed to overcome these difficulties.

The solution reported in this paper relies on the following basis, considering that Web services can be described using different languages, there are essential common elements that all Web services description languages must provide: a general communication interface that the client uses to create a proxy object to invoke the service remotely. This communication interface must describe information about the functions that the service offers (operation in WSDL, profile in OWL-S or capability in WSMO) as well as the correct description of input and output parameters. Taking into account these common elements between services described in any of these languages and that ontologies represent the cutting edge technological movement, this article describes an ontological representation for public Web services. With this motivation in mind, this paper describes two contributions:

A service mining process which extracts relevant service elements coded into service descriptions, calculates similarity measures, and discovers semantic relationships between them. In this work, the discovery of similarity relations is based on a set of structural and syntactical similarity measures.

An ontology-based representation of public Web services, which serves as a service repository which allows the dynamic acquisition of more service instances and discovery of similarities among them. This service ontology allows the definition of query rules to support complex service tasks such as search, discovery, selection, substitution and composition. An additional benefit of using an ontology-based representation of discovered similarities between Web services is the possibility of inter-connection and inter-operation with existing semantic models.

The rest of the paper is organized as follows: in Section 2, the Web service mining approach is described; in Section 3, experimentation is presented; in Section 4, useful application scenarios are described to show the applicability of this work; finally, in Section 5, conclusions are presented.

II. RELATED WORK

Web service automatic mining is the task of searching (by means of crawlers), retrieving and parsing public Web services. Research topics related with Web service mining are data mining and knowledge discovery. Hamel et al. [9] describe Web service mining as the process of applying data mining techniques on Web service logs, with the objective of discovering actionable Web service intelligence. In particular authors analyze the requirements to deal with four mining levels: the interface level, the abstract process level, the choreography level and the orchestration model of a composite Web service level. However, this work reports this analysis result with no experimentation or real implementations of the four mining levels. Among the main difficulties that they faced is “the lack of existing public Web service execution logs” to work with.

The work reported in Chen et al. [10] is closely related with this one, they use a bottom up discovery approach for mining Web services, use an ontology to represent discovered relations and a set of rules to obtain more definitions between Web services. However, they do not provide any experimental evidence of "real world" Web services.

A service mining framework is reported by Zheng and Bougettaya in [11]. In this work, authors describe a bottom up approach framework for mining Web services. Their main focus is on discovering any interesting and useful service composition that may came up during the mining process, with no goal containing specific search criteria. In contrast, the work reported in current paper has a similar intention mining process, in the sense that no objective criteria is provided, but the semantic relations discovered are not tailored only for composition purposes.

Zhang et al. [12] report a composite Web services discovery technique based on community mining. In this work, authors address the problem of finding services that are more suitable for a common goal to complete a task. In particular, they propose a method of mining the service community by exploiting service execution logs. The main difference between Zhang work and the approach described in current paper is the information source of the mining process.

Young-Ju et al. [13] argue that there exist a vast number of public available services that cannot be utilized by tools that enable service users to create mashups without programming knowledge. They propose a solution based on the combination of existing description languages and learning ontology mechanisms in order to enable the development of semantic web services compliant with architectural style of RESTful web services. Their ontology learning mechanism is used to extract and cluster service parameters, producing a parameter-based domain ontology which is evaluated against a traditional keyword-based service search mechanism.

Yousefipour et al. [14] propose an ontology-based framework for the discovery of semantic Web services (SWS) using a QoS approach. Their framework describes an ontology manager component which handles the provider

 ISBN: 978-1-61208-266-0

and the requester domain or general ontologies. This component merges these ontologies with general ontologies and creates a new generalized ontology, which is used for ranking the resulting list of SWS. Even though authors address automatic discovery of SWS by means of ontologies, their ontologies are domain-oriented. In contrast, in this paper ontologies are used for modeling service programmatic interfaces aiming at supporting automatic search and discovery of public available Web services.

Yoo Jung et al. [15] address the problem of annotating Web services from the Deep Web. Deep Web refers to Web pages that are not accessible to search engines. In particular, authors consider Web forms interface pages as Deep Web services that reflect the real content types of the Deep Web. Their proposed solution consists of the automatic generation of a domain ontology for semantic annotation of Web services. Such domain ontology is built based on Web page attributes (any items of descriptive information about the site). Their research main goal is improving the automatic search and discovery of public Web services. However, their service description sources are different as they are using Web form interface pages instead of a formal service description language.

Sabou and Pan [16] presented a study of the major problems with Web service repositories (some of them are no longer available, however the result of the study is still relevant). They concluded that Web service repositories use simple techniques of accessing the content of Web services, browsing across services listings relies on few and low quality metadata, and metadata is not fully exploited for presentation. Authors also proposed various semantic-based solutions to enhance semantically service repositories. Retaking the early ideas of these authors, the solution that is reported in this article is to lay the foundations for the automatic construction of public Web services repositories based on ontologies.

Comparison with related work. The idea of retrieving , clustering and mining Web services using a semantic approach is no new. However, none of reported works have fully achieved the level of automation with real existing Web services. Some works do not offer experimentation with real world service implementations [9], [10]. Service mining-related works were not designed with a global vision (considering programmatic interfaces and application domains) to support all service tasks; for instance [11] describes an approach tailored only for composition, [13] presents the construction of ontologies with a specific parameter-domain approach, or application-domain ontologies [14]. The rest of work use different service information sources: service execution logs [12], and Web form interface pages instead of a formal service description language [15]. The approach reported in this paper aims at automating mining real world Web services to support all service tasks based on programmatic interface ontologies and domain ontologies using as a source public Web service descriptions.

III. MINING PUBLIC WEB SERVICES

In this paper, Mining Web Services is defined as the task of unveiling similarities that hold across multiple Web services descriptions; in order to support complex tasks, such as: discovery, selection, matchmaking, substitution and composition of Web services. Data and text mining have the main purpose of discovering patterns from data and produce new information or knowledge. In this context, the objective of mining public Web services descriptions is to find certain patterns or relationships (patterns and relationships are used interchangeable as synonyms) based on a set of similarity measures.

The process of mining Web services is depicted in Figure 1. This process involves the following phases:

Retrieving public Web services. This phase consists of searching and copying service descriptions files from the Web. The objective is to gather information about services available on the Web and maintain a local repository of retrieved public Web services. To achieve this objective, a common strategy is to program several softbots that seek for services on the entire Web; or visit specific service repositories that manage lists of services. One of the most representative public Web service repositories is Seekda [17] search engine, which currently catalogs more than 28,000 service descriptions.

Parsing Web services descriptions. This phase consist of reading description files, identify the relevant elements, retrieve and process them. With this regard, two important requirements have to be addressed: heterogeneity of service description languages (SDL) and selection of the relevant data to be retrieved from service files. The former requirement is derived from the existence of various SDLs: WSDL, OWL-S, WSML, and SAWSDL. Both requirements are addressed by identifying the essential common elements that all SDLs must provide: a communication interface that the client uses to create a proxy object to invoke the service remotely. This communication interface must describe information about the functions that the service offers (operation in WSDL, profile in OWL-S or capability in WSML) as well as the descriptions (name and data types) of input and output parameters.

Similarities discovery, during this phase a set of similarity measures are calculated to enable the generation of new relationships between services and entail new knowledge about these relations. During this phase the mining process can be configured through the combination of various similarity measures in order to find more interesting results depending on the user needs and application objective.

Inference and maintainability. This phase consist of the execution of a set of inference rules which generate and maintain similarity relations into the ontology whenever new service instances are added. This phase also allows the definition of more rules to construct interesting relations based on the basic similarity relations. For instance, if there are two service functions which hold input and output parameter similarities, and also hold a semantic similarity on

 ISBN: 978-1-61208-266-0

their function names, then a combined similarity can be defined to establish a structural similarity (covering parameter and function names). Another possibility is to offer mechanisms of dependency checking if a correlated service instance is deleted.

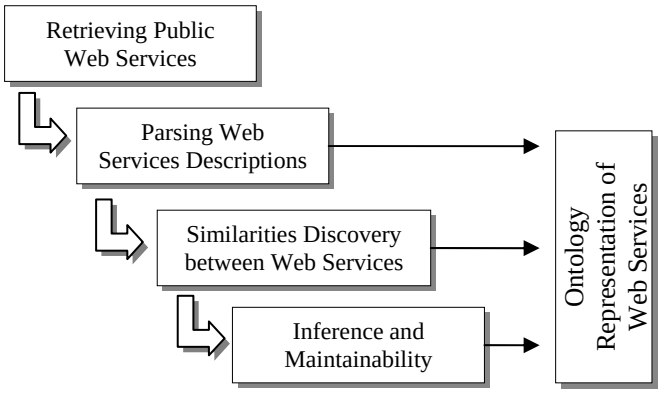

Figure 1. The process of mining public Web services.

An important element of this mining process is the ontological representation, which consists of an ontology management interface and the resulting ontology. The Ontology management is a programming interface through which the mining phases read, write and update concepts and semantic relations into the ontology.

The Web service Ontology is a formal and logical representation of the mined services together with all semantic relationships.

IV. ONTOLOGY TO REPRESENT WEB SERVICES

In 1993, Gruber defined Ontology as "an explicit specification of a conceptualization" [18]. The Web service ontology described in this paper, aims at providing fundamental inter-relations representation of Web service core concepts (functions, input parameters and output parameters). The main entities (classes) are *Service*, *Function* and *Parameter*. Figure 2 shows a general view of this ontology and its interrelationships. The *Parameter* class is sub-classified into *InputParameter* and *OutputParameter* classes.

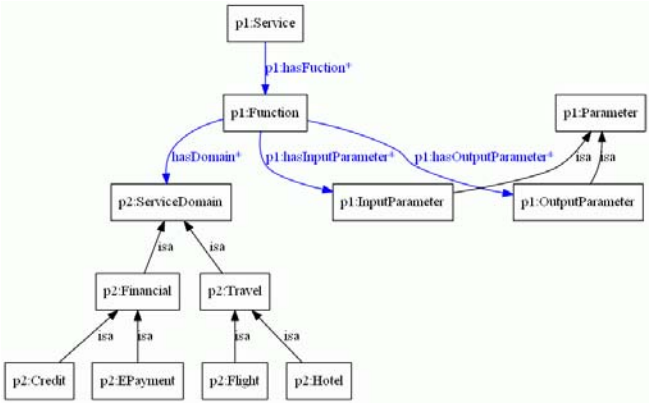

Figure 2. Ontology for the representation of Web services and their related application domains.

Data type properties were defined as follows. For class ***Service*** *hasServiceName* and *hasURL* data type properties were defined to take only *xsd*:*string* data values. For class ***Function***, the *hasFunctionName* data type property was established, allowing only *xsd:string* data values. For class ***Parameter***, the *hasParameterDataType* and *hasParameterName* data type properties were defined to take *xsd:string* data values. Identification of semantic relationships between individuals of different classes in the ontology is implemented as object properties. *Service* class relates with *Function* class through *hasFunction* object property. Restrictions to this property are that a service has at least one function, and can have many functions. *Function* class is related with class *Parameter* through the *hasInputParameter* and *hasOutputParameter* object properties. Object property *hasInputParameter* is restricted to take values only from the class *InputParameters*; likewise, *hasOutputParameters* object property takes values only from the *OutputParameters* class.

V. SIMILARITY MEASURES

Based on the work reported in Bravo and Alvarado [19], in this section various structural and syntactic similarity measures are described.

A. Function name similarity

Let $Oname_1$ and $Oname_2$ be two compound function names from two different Web services. $Oname_1$ consisting of a set of lexical tokens identified by $OnameTokens_1$. $Oname_2$ consisting of a set of lexical tokens identified by $OnameTokens_2$.

The lexical similarity between names is calculated using the Jaccard Similarity Coefficient:

$$\textbf{\textit{FunctionNameSim}}(Oname_1, Oname_2) = \left| OnameTokens_1 \cap OnameTokens_2 \right| / \left| OnameTokens_1 \cup OnameTokens_2 \right| \quad (1)$$

The ***FunctionNameSim*** similarity measure will return a value in the range [0, 1], where a returned value of 1 represents a total similarity between both function names, and a returned value of 0 represents a total difference between names.

B. Input parameter similarity

Let $O_1 = (Oname_1, Ip_1)$, $O_2 = (Oname_2, Ip_2)$ be two functions from different Web services, with $Oname_i$ representing the function name and Ip_i the set of n input parameters described as follows:

$Ip_1 = \{ (nameP_1, typeP_1), (nameP_2, typeP_2), \ldots, (nameP_n, typeP_n)\}$,

$Ip_2 = \{ (nameP_1, typeP_1), (nameP_2, typeP_2), \ldots, (nameP_n, typeP_n)\}$.

Each parameter is defined by a pair of name and data type (*nameP*, *typeP*). The input parameter similarity is calculated as follows:

$$\textbf{\textit{InputParSim}}(O_1, O_2) = \left| Ip_1 \cap Ip_2 \right| / \left| Ip_1 \cup Ip_2 \right| \quad (2)$$

The ***InputParamSim*** measure will return a value in the range [0, 1], where a returned value of 1 represents a total similarity, and a value of 0 represents a total difference.

C. Output parameter similarity

Similarly to (1), a measure to evaluate the lexical similarity between output parameter **names** is defined. Let $OPname_1$, $OPname_2$, be two output parameter names from different Web service functions, each consisting of a set of lexical tokens identified by $OPnameTokens_1$ and $OPnameTokens_2$, respectively. The output parameter name lexical similarity is calculated by:

$$\textbf{\textit{OPnameSim}}(OPname_1 , OPname_2) = \left| OPnameTokens_1 \cap OPnameTokens_2 \right| / \left| OPnameTokens_1 \cup OPnameTokens_2 \right| \quad (3)$$

The ***OPnameSim*** measure will return a value in the range [0, 1], where a returned value of 1 represents a total similarity, and a value of 0 represents a total difference.

Let $OPtype_1$, $OPtype_2$, be two output parameter data **types** from different Web service functions. The output parameter data type similarity between them is calculated as follows:

$$\textbf{\textit{OPtypeSim}}(OPtype_1 , OPtype_2) = \begin{cases} 1, & \text{if } OPtype_1 = OPtype_2 \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

The ***OPtypeSim*** measure will return a value of 1 if both types are equal, and a value of 0 if they are different.

D. Average output similarity

Let $O_1 = (Oname_1, Op_1)$, and $O_2, = (Oname_2, Op_2)$, be two functions from different Web services, with name $Oname_i$ and the output parameter object Op_i of function *i*. Each output parameter object *Opi* consists of a pair of name and data type, $Op_1 = (OPname_1, OPtype_1)$, and $Op_2 = (OPname_2, OPtype_2)$. Particularly, O_1 and O_2 are output equivalent if $((OPname_1 = OPname_2)$ and $(OPtype_1 = OPtype_2))$.

The output parameter similarity is calculated as the average of output parameter name similarity and output parameter data type similarity as follows:

$$\textbf{\textit{OutputParSim}}(O_1 , O_2) = [OPnameSim(OPname_1 , OPname_2) + OPtypeSim(OPtype_1 , OPtype_2)] / 2 \quad (5)$$

The ***OutputParSim*** measure will return a value in the range [0, 1], where a returned value of 1 represents a total similarity, and a value of 0 represents a total difference.

E. Structural similarity

Structural similarity represents the average of parameter name similarity (1), input parameter similarity (2) and output parameter similarity (5). Let $O_1 = (Oname_1, Ip_1, Op_1)$, and $O_2 = (Oname_2, Ip_2, Op_2)$, be two Web service functions with their respective sets of input and output parameters; the level of structural similarity between them is calculated as follows:

$$\textbf{\textit{StructuralSim}}(O_1 , O_2) = [\textbf{\textit{FunctionNameSim}} (Oname_1 , Oname_2) + \textbf{\textit{InputParSim}}(O_1 , O_2) + \textbf{\textit{OutputParSim}}(O_1 , O_2)] / 3 \quad (6)$$

The ***StructuralSim*** measure will return a value in the range [0, 1], where a returned value of 1 represents a total similarity, and a value of 0 represents a total difference.

Further similarity measures can be defined and combined to obtain more interesting similarity results between services.

VI. EXPERIMENTATION

For experimentation 37 public Web service descriptions (WSDL) files were retrieved from Seekda [17]. The architecture depicted in Figure 1 was implemented as follows: a Web service data extraction module, which browses any set of public available WSDL files and extracts the service name, the set of function names, the names and data types of input and output parameters; an ontology population module, which registers into the ontology new function instances after data is extracted from WDSL files, and a similarity relations discovery module, which calculates structural similarities between function pairs and registers new semantic relations between compared individuals, if the level of similarity resulted higher than a threshold.

The parser module extracted the function names, input and output parameter names and types from the initial 37 Web service description files. As a result, the ontology was populated with a total of 537 new *Functions*, and 6317 *Parameters*: 3155 *InputParameters* and 3162 *OutputParameters*.

The discovery of structural similarity relations was calculated between all individuals from the *Function* class. Resulting relations were named *isFunctionNameSimilarTo*, *isOutputParamSimilarTo*, *isInputParamSimilarTo*, and *isStructuralSimilarTo*. If the resulting level of similarity is higher than an established threshold, then a similarity relationship is generated between both functions in the ontology. For the set of 537 functions, similarity results are shown in Table 1. Thereafter, the ontology continues growing in similarity relationships as new services are registered. In this case, the resulting ontology is considered dynamic and evolving over time.

TABLE I. NUMBER OF DISCOVERED RELATIONS

Similarity relationship	**Total**
Function name similarity	213
Input parameter similarity	470
Output parameter similarity	1440
Structural similarity	184

The set of structural similar relationships is a combined result of the function name, input parameter and output parameter similarities for each function pair, which is the main reason of the reduced number of relations in comparison with the three previous.

VII. APPLICATION CASES

Searching and discovery of specific service functionalities are among the most important service-related tasks, because it allows software developers and integrators to find specific services functionalities which satisfy their needs. Majority of Web service repositories offer basic search mechanisms, mostly based on key-word and service category matching. The service ontology reported in this paper supports the same search mechanism, but the set of similarity relations discovered and established between services functions; allow seeking and finding more services functions that are structurally related, returning more and significant functions. The following rules are specified to query the ontology and obtain answers about the set of functions being treated. The query rule showed in (7) displays pairs of *Functions* instances for which a relation of *Input Parameter* similarity was discovered and established.

Function(?*x*) ∧ *Function*(?*y*) ∧ (7)
isInputParamSimilarTo(?*x*, ?*y*)
→ *sqwrl:select*(?*x*, ?*y*)

Similarly, the query rule shown in (8) displays pairs of *Functions* instances for which a relation of *Output Parameter* similarity was discovered and established.

Function(?*x*) ∧ *Function*(?*y*) ∧ (8)
isOutputParamSimilarTo(?*x*, ?*y*)
→ *sqwrl:select*(?*x*, ?*y*)

The query rule showed in (9) displays pairs of Functions individuals for which a relation of Function Name similarity was discovered and established.

Function(?*x*) ∧ *Function*(?*y*) ∧ (9)
isFunctionNameSimilarTo(?*x*, ?*y*)
→ *sqwrl:select*(?*x*, ?*y*)

Finally, the query rule presented in (10) is very useful because it allows inferring what functions may be ***substitutable*** each other, provided they meet three conditions: Input Parameter similarity, Output Parameter similarity and Function Name similarity.

Function(?*x*) ∧*Function*(?*y*) ∧ (10)
isInputParamSimilarTo(?*x*, ?*y*) ∧
isOutputParamSimilarTo(?*x*, ?*y*) ∧
isFunctionNameSimilarTo(?*x*, ?*y*)
→ *sqwrl:select*(?*x*, ?*y*)

Rule (11) searches flight service functions which return flying routes. Results of this query-rule are shown in Table 2.

FlightServices(?*x*) ∧ (11)
hasOperation(?*x*, ?*y*) ∧
hasOperationName(?*y*, ?*str*) ∧
swrlb:contains(?*str*, "*Route*")
→ *sqwrl:select*(?*x*, ?*y*)

TABLE II. SERVICES THAT OFFER FLYING ROUTES FUNCTIONS

Service	Function
Volagratis	Volagratis-getRoutes
Arc	Arc-GetRoutes

Using the service ontology it is possible to extend the search of flight functions using the input parameter similarity relation. Rule (12) searches flight service functions which return flying routes and similar functions which hold an *Input Similarity* relationship. Result is shown in Table 3.

Function(?*x*) ∧ (12)
hasOperationName(?*x*, ?str) ∧
swrlb:contains(?str, "Route") ∧
Functions(?*y*) ∧
isInputParamSimilarTo(?*x*, ?*y*)
→ *sqwrl:select*(?*x*, ?*y*)

TABLE III. SERVICES THAT OFFER FLYING ROUTES FUNCTIONS

Function	Function
Arc-GetRoutes	Arc-GetAvailability

Another example of searching "booking" functions using the function name similarity relation is executed with the query rule (13). Results of this query are shown in Table 4.

Function(?x) ⋀ (13)
hasOperationName(?x, ?str) ⋀
swrlb:containsIgnoreCase(?str, "booking") ⋀
Function(?y) ⋀
isOperationNameSimilarTo(?x, ?y)
→ *sqwrl:select*(?x, ?y)

TABLE IV. SERVICES THAT OFFER FLYING ROUTES FUNCTIONS

Function	Function
Hotelmercado_WS-SetConfirmBooking	Hotelmercado_WS-ConfirmBooking
Hotelmercado_WS-GetBookingInfo	Hotelmercado_WS-GetProcBookingInfo
TourConexWebService-doMainServiceHotelBooking	TourConexWebService-doMainServiceCarBooking
TourConexWebService-doBookingStatistic	TourConexWebService-doBooking
pegas-cancelBooking	MORSWebService-CancelTransferBooking
pegas-cancelBooking	MORSWebService-CancelHotelBooking
TourConexWebService-doCarBooking	TourConexWebService-doMainServiceCarBooking
TourConexWebService-doCarBooking	TourConexWebService-doBooking
pegas-confirmBooking	Hotelmercado_WS-SetConfirmBooking
pegas-confirmBooking	Hotelmercado_WS-ConfirmBooking
TourConexWebService-doBooking	TourConexWebService-doErvBooking
TourConexWebService-doBooking	TourConexWebService-doTicketsafeBooking
TourConexWebService-doBooking	TourConexWebService-doMultiBooking

Substitution is another important task for the Web service community; it allows searching and selecting a similar service function that matches input and output parameters. The query-rule (14) applied to the service ontology helps the developer to search and find "substitutable" services, based on syntactic and structural similarity measures. Results of

 ISBN: 978-1-61208-266-0

this query are shown in Table 5. A normal and common service repository does not support such kind of searches. The developer should do so manually, requiring more effort and time.

Function(?*x*) ∧ (14)
Function(?*y*) ∧
isInputParamSimilarTo(?*x*, ?*y*) ∧
isOutputParamSimilarTo(?*x*, ?*y*) ∧
isOperationNameSimilarTo(?*x*, ?*y*) ∧
hasOperationName(?*x*, ?*str1*) ∧
hasOperationName(?*y*, ?*str2*) ∧
swrlb:notEqual(?*str1*, ?*str2*)
→ *sqwrl:select*(?*x*, ?*y*)

TABLE V. SUBSTITUTABLE SERVICE FUNCTIONS

Function	Function
BookingLand-CountryProvinceList	BookingLand-CountryProvinceCityList
CHotelsWebService5-startTransaction	CHotelsWebService5-startTransactionMulti
WSNewHotelSrv-GetSimpleAvailability	WSNewHotelSrv-GetSimpleAvailabilityTeste
MORSWebService-Ping	CreditCardServiceV1-ping
WSNewHotelSrv-MakeSimpleReservationTeste	WSNewHotelSrv-MakeSimpleReservation
pegas-getSpecifiedFlightList	pegas-getFlightList
WSNewHotelSrv-DeleteUserHotel	WSNewHotelSrv-DeleteHotel
WSNewHotelSrv-DeleteUserHotel	WSNewHotelSrv-DeleteUser
BookingLand-ProviderAvailabilityEx	BookingLand-ProviderAvailability
BookingLand-ProviderSearchQuick	BookingLand-ProviderSearchQuickEx

VIII. PERFORMANCE ANALYSIS

Performance analysis of service-related tasks is an important issue whenever these tasks are based on ontological representation. In particular, in this paper the following service tasks are of performance concern:

Ontology population. This is the most time consuming task because for each service instance treated requires the execution of two operations: *service parsing* and *service ontology recording*. Which means that for each service, the parser extracts its operation names and respective input and output parameters, and then records all instances into their ontology classes. In particular, for the set of 37 initial *Services* used for experimentation, a total of 537 *Functions*, and 6317 *Parameters* were registered into the ontology file. Therefore, this task required a total of 37 service parsing operations and the sum of 37 + 537 + 6317 = 6891 ontology-write operations. Obviously, the more service instances are treated the more time is needed. However, this particular time-consuming task is not considered as critical, because it is executed only once per service set. Even more, when new service instances are to be recorded into the same ontology, they are first validated for non redundancy, therefore only new different services are allocated. Ontology population is a time-consuming task, but is not a frequent task.

Search, discovery and substitution. In a traditional implementation approach these tasks would require traversing the entire ontology T-Box and A-Box to find particular class instances, relation instances or individuals. However, in this paper the use of a rule language enhanced with querying constructs (SWRL) allows the definition and execution of rule-based search, discovery and substitution. A rule-based querying mechanism offers improved performance, as it filters only the necessary class, relations, axioms and individuals needed for the execution of each rule. For instance, when the inference engine executes the query rule (7) it requires to load a total of 537 *Function* instances and 470 *InputParameterSimilarity* relations, resulting in a space-reduced selection operation.

IX. CONCLUSIONS AND FUTURE WORK

Results show advances on automatic similarity relations discovery from public available Web service descriptions. The automatic population of the ontology with existing WSDL files is a relevant advance towards the automated reutilization and construction of service-based solutions using pre-existing resources. Resulting similarities between service functions show that the set of measures calculations can be combined to obtain more complex and significant information concerning functions inter-relations. This combination of measures can be conducted by implementing more similarity methods or by defining additional rules of inference. This process can be defined as Web services mining in the sense that helps to discover unknown relationships between functions. Inference is a key issue for maintainability and evolution of the ontology; inference rules generate new inter-relationships between functions and help to answer constrained queries regarding asserted inter-relationships in the ontology. Experimental results show that the overall process towards the automation of public Web services mining based on ontology population and structural similarity measures is feasible and can be completely automated.

The next steps of this research are the implementation and combination of more sophisticated similarity measures to facilitate automatic discovery and composition of Web services. In particular, behavior similarity measures, data type comparison measures, and linguistic patterns will be designed and applied to discover deeper semantic relations between public available Web services.

ACKNOWLEDGEMENTS

Authors thank the ICYTDF and UAM for their financial support granted for the presentation of this research.

REFERENCES

[1] http://www.w3.org/TR/wsdl, last visited 15.03.2013.

[2] http://uddi.org/pubs/uddi_v3.htm, last visited 15.03.2013.

[3] http://www.ebxml.org, last visited 15.03.2013.

[4] M. Rodríguez, M. Crasso, A. Zunino, and M. Campo, "Improving Web Service descriptions for effective service discovery," Science of Computer Programming, vol. 75, no. 11, Elsevier Science, 2010, pp. 1001-1021, doi: 10.1016/j.scico.2010.01.002.

[5] S. McIlarith, T. Cao Son, and H. Zeng, "Semantic Web Services," IEEE Intelligent Systems, vol. 16, no. 2, IEEE, 2001, pp. 46-53.

[6] http://www.w3.org/Submission/OWL-S, last visited 15.03.2013.

 ISBN: 978-1-61208-266-0

[7] http://www.wsmo.org, last visited 15.03.2013.

[8] http://www.w3.org/TR/sawsdl, last visited 15.03.2013.

[9] L. Hamel, M. Graiet, M. Kmimech, M. Bhiri, and W. Gaaloul, "Verifying Composite Service Transactional Behavior with EVENT-B," Proc. of the 5th European Conference on Software Architecture (ECSA 2011), Springer LNCS, September 2011, pp. 67-74, doi: 10.1007/978-3-642-23798-0.

[10] S. Chen, Z. Feng, H. Wang, and T. Wan, "Building the Semantic Relations-Based Web Services Registry through Services Mining," Proc. of the Eigth IEEE/ACIS International Conference on Computer and Information Science (ICIS 2009), IEEE, June 2009, pp. 736-743, doi: 10.1109/ICIS.2009.20.

[11] G. Zheng and A. Bouguettaya, "A. Service Mining on the Web," IEEE Transactions on Services Computing, vol. 2 no. 1, IEEE, January-March 2009, pp. 65-78, doi: 10.1109/TSC.2009.2.

[12] X. Zhang, Y. Ying, M. Zhang, and B. Zhang, "A Composite Web Services Discovery Technique Based on Community Mining," Proc. of the IEEE Asia-Pacific Services Computing Conference (IEEE APSCC), IEEE, December 2009, pp. 445-450, doi: 10.1109/APSCC.2009.5394087.

[13] L. Young-Ju and K. Chang-Su, "A Learning Ontology Method for RESTful Semantic Web Services," Proc. of the International Conference on Web Services (ICWS 2011), IEEE, July 2011, pp. 251-258, doi: 10.1109/ICWS.2011.59.

[14] A. Yousefipour, M. Mohsenzadeh, A. Ghari, and M. Sadegzadeh, "An Ontology-based Approach for Ranking Suggested Semantic Web Services," Proc. of the 6th International Conference on Advanced Information Management and Service (IMS 2010), IEEE, December 2010, pp. 17-22.

[15] Y. An, J. Geller, Y. Wu, and S. Ae Chun, "Automatic Generation of Ontology from the Deep Web," Proc. of the 18th International Workshop on Database and Expert Systems Applications (DEXA 2007), IEEE, September 2007, pp. 470-474, doi: 10.1109/DEXA.2007.43.

[16] M. Sabou and J. Pan, "Towards semantically enhanced Web service repositories," Journal of Web Semantics: Science, Services and Agents on the World Wide Web, vol. 5, no. 2, June 2007, pp. 142-150, doi: 10.1016/j.websem.2006.11.004.

[17] http://www.seekda.com, last visited 15.03.2013.

[18] T. Gruber, "A Transaltion approach to portable ontologies," Knowledge Acquisition, vol. 5, no. 2, June 1993, pp. 199-220, doi: 10.1006/knac.1993.1008.

[19] M. Bravo and M. Alvarado, "Similarity Measures for Substituting Web Services," International Journal of Web Services Research, vol. 7, no. 3, July 2010, pp. 1-29, doi: 10.4018/jwsr.2010070101.

 ISBN: 978-1-61208-266-0