SERVICE ACQUISITION AND VALIDATION IN A DISTRIBUTED SERVICE DISCOVERY SYSTEM CONSISTING OF DOMAIN- SPECIFIC SUB-SYSTEMS

Deniz Canturk and Pinar Senkul
METU, Computer Engineering Department, 06531 Ankara, Turkey

Keywords: Distributed, Domain specific and Semantic web service discovery, Ontology, Quality of Service.

Abstract: With the increase in the number of advertised web services, finding the web services appropriate for a given request will result in problems in terms of performance, efficiency and quality of the discovered services. In this paper, we introduce a distributed web service discovery system which consists of domain-specific sub-systems. We establish ontology based domain specific web service discovery sub-systems in order to discover web services on private sites and in business registries, and in order to improve service discovery in terms of efficiency and effectiveness. An important problem with effectiveness is to check whether a discovered service is active or not. Many services in the registries are not active any more. Distributing the discovery to domain-specific subsystems makes it possible to lower the service discovery duration and to provide almost up-to-date service status. In this work, we describe the service acquisition and validation steps of the system in more detail and we present experimental results on real services acquired on the web.

1 INTRODUCTION

Web services begin to play important role in Web computing. However, finding the most appropriate Web service among the vast amount of advertised services becomes a hard task. Different approaches with specific designs and realization are introduced to overcome this problem. However these proposed approaches have some drawbacks.

Most of the Web service discovery approaches are based on syntactic matching between the service advertisement and the service query. Popular service registries such as UDDI (Universal Description, Discovery and Integration) (uddi, 2004) and ebXML (Electronic Business using XML) (ebXML, 2005) provide ways for locating businesses, however, they are limited to keyword-based search techniques. In addition, they do not provide QoS information and service lifecycle tracking since they are disconnected from the services they advertise.

Most of the proposed systems are not scalable, i.e. they have difficulty in handling the large number of service providers and service requesters. This is basically due to that fact, Web services are collected in a central registry or central service databases. When the database becomes overloaded, scalability problem is resolved by using replica of central registry of database. However replica management consumes high operational and management expenses.

Another important drawback is that current solutions are incompatible with each other. Existence of heterogeneous service registries (UDDI, ebXML), private service portals and discovery approaches (SPIDeR (Sahin, Gerede, Agrawal, Abbadi, Ibarra and Su, 2005) or METEOR-S (Meteor, 2006)) makes it complicated to publish and search a service. Users spend too much time to visit numerous user interfaces, to understand the way to use them, to enter query text again and again, and assembly the results from each interface.

Finding the most appropriate Web service for user's request manually is a tedious task under these conditions. Although there are many services being published, they are scattered on the Web and there is not a feasible solution yet to bring them together. This problem is similar to information search issue, which is handled by search engines (such as Google, Yahoo, etc.) nevertheless there is no such established solution for Web service querying.

Therefore, we aim to develop such a solution and eliminate the burden of facing different web service registries, by using a single search interface with semantic and QoS querying capability. However, such a system should not have a centralized structure in order to avoid efficiency and scalability problems. For this reason, we propose a service discovery system that uses Domain-Specific Web Service Discoverer (DSWSD-S) sub-systems which are built around ontology. Since a (DSWSD-S) has its own compact ontology, shorter update time and querying is provided. This structure also supports scalability. The proposed architecture supports QoS aware service discovery with syntactic and semantic matching capabilities while not disturbing the autonomy of web service discovery. In addition, QoS information can be collected automatically. Construction of a DSWDS-S provides a framework for semantic annotation of available web services, as well.

Acquisition of the web services is the first step of a DSWSD-S. In this step, the services that are related with a given ontology are extracted and analyzed. It includes the analysis of parameters as well. In this work, the emphasis is on the acquisition step of DSWSD-S.

The rest of this paper is organized as follows: In Section 2, we describe the DSWSD-S architecture. In Section 3, the design and implementation of service acquisition and validation in DSWSD-S is presented. Section 4 contains the experimental results on the acquisition and validation of real web services. Related work is given in Section 5. Finally, conclusion is presented in Section 6.

2 DISTRIBUTED SERVICE DISCOVERY SYSTEM CONSISTING OF DOMAIN- SPECIFIC SUB-SYSTEMS

2.1 Overall Architecture

The proposed system consists of domain-specific subsystems, each of which is specialized for an ontology, that is, each subsystem crawls, discovers and indexes web services according to its own ontology. Main idea in the proposed system is to split the discovery task into subtasks on the basis of the domains specified with an ontology, in order to provide parallel processing and scalability. In addition to these advantages, it facilitates load balancing in domain-specific web service database generation and collaboration between DSWSD-Ss in web service querying. Crawling process continues in a cycle in order to keep the web service database up-to-date. As the crawling cycle gets shorter, web service suggestion becomes more reliable. The distributed structure supports shorter crawling cycles by distributing the load to subsystems.

The overall structure is depicted in Figure 1. The architecture for DSWSD-S consists of two layers: domain-specific crawler layer and domain-specific service discovery layer.

2.1.1 Domain-specific Crawler Layer

Web service database generation according to its own ontology is the main task of a domain-specifi crawler layer. Web service database generation process starts with URL and service description acquisition of advertised Web services. In order to collect the descriptions, a focused crawling is performed based the ontology of DSWSD-S. The next step is to check whether the acquired web services are still active and accessible or not, by calling simple yet appropriate input parameters.

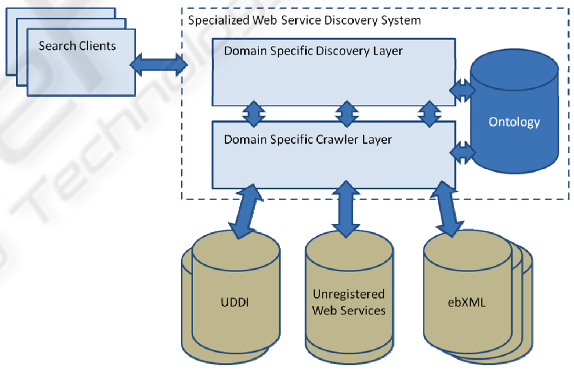
Search Clients
Specialized Web Service Discovery System
Domain Specific Discovery Layer
Ontology
Domain Specific Crawler Layer
UDDI
Unregistered Web Services
ebXML

Figure 1: Distributed Service Discovery Architecture.

In service extraction phase, validated web service's input and output parameters are compared with the ontology terms in order to determine the ontological position of the web service. Both syntactic and semantic matching is used in this phase. Semantic matching provides better results than syntactic matching. However semantic matching requires input and output parameters of the web service to be annotated by using the DSWSD-S ontology. Considering that the most of the available web services do not have semantic annotation, DSWDS-S supports semantic matching as well.

In service verification phase, extracted services are verified whether web services actually perform the advertised tasks. This is a hard and time consuming process since (i) what a service provides should be automatically discovered, (ii) appropriate inputs should be set, and (iii) it should be determined that the appropriate outputs are obtained for the given inputs. If the inputs and outputs are semantically annotated, service verification can be fulfilled with the help of ontology. Otherwise, input and output parameters are compared with the ontology terms to find the matching degree of parameter to the DSWSD-S ontology.

In quality of service (QoS) determination step, web service is evaluated for non-functional attributes such as response time, throughput, reliability and availability. In order to obtain QoS values automatically, the information obtained in validation phase is recorded and analyzed.

2.1.2 Domain-Specific Service Discovery Layer

This layer provides a user interface to query the ontology specific Web service database. The provided information can be both in syntactic and semantic form. User can also set QoS constraints for the desired Web service.

As the first step, keywords provided by the user are compared with the ontology terms in order to figure out the ontological position of the required service. When the keyword matching value is above the confidence level, its ontological children terms are also added to search set. If the overall matching value is above the confidence level, web discovery is performed in the database of the initially accessed DSWSD-S. Otherwise, request is considered as unrelated with this domain and forwarded to other domain specific web service discovery peers.

DSWSD's database is queried to discover the web services fulfilling the request. In addition to syntactic matching of entered keywords, semantic keyword sets are used in order to find semantically related web services. Fast access to database records is achieved by using inverted index. This table, which is filled in service extraction phase, includes ontology term, web service URL and matching degree columns. In service evaluation phase, web service suggestion values are determined with respect to the conformance to request and QoS values of the web service. Web services are sorted with suggestion values and best ones are presented to the user.

2.2 Network Topology among DSWSD-S Nodes

DSWSD-Ss establish peer-to-peer (P2P) network in order to communicate with each other. Custom p2p network infrastructure is implemented instead of using the p2p network implementations in literature such as Gnutella (Gnutella, n.d.) or Chord (Morris, Karger, Kaashoek and Hari Balakrishnan, 2001). Communication among DSWSD-S is handled by simple text messaging. In other words, a DSWSD-S sends XML based text message requests to the other entire peer subsystems and waits for the response messages. Hence, sophisticated features of P2P networks are not required.

Basic operations in this network structure are as follows:

Join to Network: The information of the new DSWSD-S is introduced to one of the peer DSWSD-S. Peer DSWSD-S sends the specifications of other DSWSD-Ss to the new DSWSD-S. New DSWSD-S sends an introduction message that contains its details to each of other DSWSD-Ss. Every peer is expected to send an acknowledge message to the new DSWSD-S.

Leave from Network: Any peer can leave the network in two different ways: explicitly by initiating a resign procedure or implicitly, or without informing any other peers. In resign procedure resigning DSWSD-S sends resign message to other peer DSWSD-Ss then immediately leave the network without waiting any acknowledge message. In the former case, if a DSWSD-S peer detects a down peer with down time above a previously set threshold, then it sends other peers down message about this peer. DSWSD-S peer that receiving down message updates peer routing table.

Related Node Routing: When a service receives a discovery request which is not related its own ontology, it forwards the request to the peers on the basis of the information in peer routing table and begins waiting. If there were any discovery results, results are redirected to the requester peer DSWSD-S.

3 SERVICE ACQUISITION AND VALIDATION

The basic step in service acquisition is collecting the addresses of Web services. This step is similar to job performed by search engines. They traverse the whole web and collect the necessary data. Since, in this work, the objective is not building a crawler, but rather effective service discovery, initially, we decided to use an off-the-shelf crawler and examined several existing open source crawlers, including Java-based crawlers and .net based crawlers. In order to provide a seed to the crawlers, we used the results obtained from search engines (Google and Yahoo). However, this approach could retrieve only a few web service description files. For this reason, we decided to try other ways to collect service descriptions.

As an alternative approach, we considered using search engines directly in finding Web service description sources. In this approach, we used the filtering capability of search engines of Google and Yahoo for file extensions while searching Web content. Most of the Web service descriptions are constructed in WSDL (wsdl, 2001) and stored with ".wsdl" extension. At this point, as described in the next paragraph in more detail, we made another extension. Since ".wsdl" file type filtering did not yield satisfactory enough results, we preferred to use ".asmx" in file filtering.

Web service description file in WSDL format contains many constructs. Indeed, these constructs are XML based text files. Therefore, WSDL file should be parsed in order to retrieve advertised web services provided on a given URL. To this aim, we implemented a customized WSDL parser that extracts inputs, output, documentation of web service and complex types defined as either input or output. We have observed that not all of the descriptions and well-formed and the parser return error messages for such cases. When we examined the error messages, we have seen three types of problems with the service descriptions:

- Service description can be specified in an earlier version of WSDL.
- Service description can be a malformed WSDL document.
  Service description is not given in WSDL at all; it is specified as an ordinary html document with wsdl extension.

The validation task is applied on the parsable descriptions. The next concern is to identify the input types to provide suitable values for parameters. Input types may be in primitive or complex structure. Handling the primitive typed parameters is straightforward, by using the following rules:

- For primitive types of enumeration, character, boolean, integer and floating point number, the input is set as the value1.
- Check string type parameter names are checked to see if it contains the word "date". If it contains the word "date", use the date that has the same month and day value and that is closest to the current date as the parameter value.
- For other strings, the parameter is set to be a simple text such as "text".

When compared with primitive types, handling the complex typed parameters is a challenging issue. While dealing with complex types, we have encountered the following situations that problems:

- complex type of a parameter may refer to another complex type
- parameter type is an array of complex type
- parameter type is a reference type.
- parameter type is an interface.
- parameter type is a nullable primitive.

For each of these situations, we used the following solutions:

- In order to create an instance of a complex typed parameter, *Reflection* utility of .net framework is used. Existence of a reference to a previously created type may lead to infinite loop. For this reason, constructed complex type instances are kept in a list and we refer to this list for setting the values when necessary.
- For complex typed arrays, we create one dimensional array having a single place. Its instance is constructed and set by using the procedure described above.
- When parameter is of a reference type, only the instance of top level type is created and we do not set any values to its attributes.
- In order to provide Interface type parameters, all the complex types that are previously defined are searched until. When a type implementing that interface is found is found, type instance is created and values are set.
- Since primitive types cannot be set to null due to its nature, instead of a primitive typed instance, an instance of object type that corresponds to the given primitive type is constructed and its value is set.

After solving all the problems with the complex types, web services are invoked one by one and the responds are analyzed to find the validated (active) services. For the filtering with ".wsdl" files, although a long source URL list is obtained, most of the descriptions cause parse errors. Among the parsable ones, the number of validates services is

Table 1: wsdl vs asmx in general.

	Valid WSDL Ratio	Valid Service Ratio
Car – wsdl	60 %	23 %
Car – asmx	35 %	93 %
Aeronautic - wsdl	23 %	48 %
Aeronautic - asmx	43 %	96 %

Table 2: wsdl vs asmx details.

	Car – wsdl	Car – asmx	Aeronautic – wsdl	Aeronautic – asmx
Total # of URLs	50	161	111	162
Valid # of URLs	30	58	26	71
Invalid # of ULRs	20	103	85	91
Total # of Service	772	632	82	1852
Valid # of Service	213	588	39	1786
Invalid # of Service	559	44	43	66
Duration	37 Mins.	17 Mins	36 Mins	48 Mins

quite few, showing that these services are not active and callable any more. Since filtering ".wsdl" files approach did not produce satisfactory results, considering that there must be much more Web service available then validated set and some of them can also be dynamically created by web servers from .asmx files with "wsdl" parameter as query string (---.asmx?wsdl), we decided to use ".asmx" file filtering in address acquisition phase. When the service provider address list obtained with this new filtering was evaluated, we have observed that the number parse problems decreased and the number of active services increases. The comparison of these two different file filtering attempts is presented in Section 4, Table 2.

4 EXPERIMENTAL RESULTS ON SERVICE ACQUISTION AND VALIDATION

In this section we present results of the service acquisition and validation on real web services. In these experiments, we used "*car*" and "*aeronautic*" domains for the domain-specific service crawling as described in Section 2.1.1. We used the results of single runs and each run is performed on the same computer with 2.73 GHz Core2Duo processor under the operating system Windows 7.

As described in Section 3, using different file filtering gives different performance results. In Table 1, comparison of using ".wsdl" and ".asmx" file filtering is presented. As seen in the results, the percentage of addresses obtained with ".wsdl" filtering changes with respect to the ontology. However, from the aspect of active web services, there is a consistent result showing that there is a higher number of active web services acquired with ".asmx" filtering.

More detailed results of this experiment are presented in Table 2. It is observed that for both ontology, ".asmx" file filtering acquires more service provider addresses than ".wsdl file filtering" The number of services under ".asmx" files is more than ".wsdl" files both in percentage and magnitude. As an expected result, it is observed that the runs take longer time for "wsdl" files due to existence of high number offline web services. DSWSD-s tries to connect to a web server and waits until time-out duration, if the server is offline and this waiting time increases the overall service acquisition time.

For the active services, we conducted further examination in order to understand the status of the services. The results for "wsdl" file filtering for under car domain are shown in Table 3. As a result of this examination, active services are further grouped under Successful Invocation, Suitable Parameter, Unsuitable Parameter, Unsuccessful Invocation, Argument Error and Insecure Invocation.

For the active services, we conducted further examination in order to understand the status of the services. The results for "wsdl" file filtering for under car domain are shown in Table 3. As a result of this examination, active services are further grouped under Successful Invocation, Suitable Parameter, Unsuitable Parameter, Unsuccessful Invocation, Argument Error and Insecure Invocation. Successful Invocation indicates that the

Table 3: Run Details.

Number of total URLs	50
Parsable URLs	30
Active Services	213
Successful Invocation	167
Suitable Parameter	34
Unsuitable Parameter	133
Unsuccessful Invocation	46
Argument Error	31
Insecure Invocation	15
Offline Services	559
Unparsable URLs	20

web service was invoked successfully and returned appropriate outputs. In case of unsuitable input parameters, “soapException” is obtained. For example, assume a service with the interface *getProduct(int productId).* Possibly, productId is the key of the record in provider’s database. If the value assigned to productID is a valid value in the provider’s database by coincidence, a successful invocation occurs. Otherwise, *soapException* is obtained. Unsuccessful Invocation means that web service cannot be invoked successfully although service is online. It occurs when the created parameter types are not compatible (Argument Error) or when service requires connection over https. (Insecure Invocation).

5 RELATED WORK

Web service discovery is a very active research area. Current studies are generally on Web service database construction, web service crawler implementation, semantic web service discovery, indexing of semantically annotated services, quality of service (QoS) issues and domain specific web service discovery.

Al-Masri & Mahmoud (2007a, 2007b, 2008) works on building a web service crawler. Web Service Crawler Engine (WSCE) is able to handle WSDL files and UBRs information concurrently and store the collected information in a centralized database called Web Service Storage (WSS).

Gooneratne & Tari (2007, 2008) works on semantic web service discovery. In their work, service discovery is done under functional and non-functional requirements or both.

Kuang, Ying, Wu, Deng and Wu (2007) propose indexing all ontology-annotated data in registered services. Based on indexing, composition-oriented service discovery is proposed.

Xu, Martin, Powley and Zulkernine (2007) works on QoS attributes of web services and they propose a QoS based web service discovery approach. In their work, every web service is scored according to the feedback of users about the service performance. The reputation manager collects feedback, calculates reputation scores, and updates these scores in the rating database.

Ma, Wang, Li, Xie and Liu (2008) works on QoS attributes of web services in a semantic point of view. They propose a semantic QoS-aware discovery framework that makes use of constraint programming.

Rocco, Caverlee, Liu and Critchlow (2005) propose service class driven dynamic data source discovery on domain specific web service. They search “Deep Web” which refers to the dynamic collection of Web documents that are created as a direct response to some user query. Our work differs from the abovementioned studies with its feature of being a collection of domain-specific subsystems. This improves the service acquisition and validation cycle time improving the service suggestion. In addition, the proposed work improves the previous studies on several points such as using semantic matching in querying and including matching value in the index.

6 CONCLUSIONS

In this work, we present a distributed service discovery system consisting of domain-specific service discovery subsystems. In this system, ontology driven domain specific web service discoverer sub-systems are executed in parallel to handle the acquisition of the services. By this way, service discovery process is accelerated and more accurate results can be obtained in terms of service validation, verification and QoS values.

In this paper, while the architecture of the system and main functionalities of modules and relations between modules are presented, the emphasis is on service acquisition and validation steps. These steps are important for the rest of the modules, since they provide the input to be processed. Our experimental evaluations on the service acquisition and validation show interesting results on the current web services. It has been observed that the search engines are better tools than using off-the-shelf crawlers. This may be due to nature of the seed sets used in the experiments. However, using search engines facilitates the acquisition task by removing the need for trying out different seed sets for satisfactory service acquisition. Another important observation is that currently, using .asmx files for dynamic service specification is a common practice. For this reason, ".asmx" file filtering on search engines provides better service acquisition than ".wsdl" file filtering. Another observation that supports this finding is that most of the services acquired under ".wsdl" filtering are not active any more.

REFERENCES

Al-Masri, E & Mahmoud, QH, 2007, Crawling multiple UDDI business registries, *16th WWW Conf.* Pp. 1255-1256.

Al-Masri, E & Mahmoud, QH, 2007, WSCE: A crawler engine for large-scale discovery of web services, ICWS pp. 1104-1111.

Al-Masri, E & Mahmoud, QH, 2008, Investigating web services on the world wide web., WWW pp.795-804

ebXML, 2005, Registry services specification v3.0, http://www.oasis-open.org/specs/index.php

Gnutella, viewed Jan 2010 http://rfc-gnutella. sourceforge.net/.

Gooneratne, N & Tari,Z, 2008, Matching independent global constraints for composite web services, WWW pp. 765-774.

Gooneratne, N, Tari, Z & Harland, J, 2007, Verification of Web Service Descriptions using Graph-based Traversal Algorithms. In *Proceedings of the ACM Symposium on Applied Computing*, pages 1385–1392.

Kuang, L, Li, Y, Wu, J, Deng, SG & Wu, Z, 2007, Inverted Indexing for Composition-Oriented Service Discovery. ICWS pp.257-264.

Ma, Q, Wang, H, Li, Y, Xie, G & Liu, F, 2007, A Semantic QoS-Aware Discovery Framework for Web Services. ICWS pp.129-136

METEOR-S: 2006, viewed Jan 2010, http:// lsdis.cs.uga.edu/projects/ meteor-s/

Morris, R, Karger, D, Kaashoek MF & Balakrishnan, H, 2001, Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applicaons, Ion Stoica

Rocco, D.; Caverlee, J.; Liu, L.; Critchlow, T. 2005, Domain-specific Web service discovery with service class descriptions, ICWS. pp.481 - 488

Sahin, OD, Gerede, CE, Agrawal, D, Abbadi AE, Ibarra, OH & Su, J, 2005 Spider: P2p-based web service discovery. *3rd International Conference Service-Oriented Computing* (ICSOC 2005), pages 157–169. Springer

UDDI, 2004 Version 3.0.2, viewed Jan 2010, descriptions http://uddi.org/pubs/uddi_v3.htm.

WSDL, 2001, version 1.1, viewed Jan 2010, descriptions Mar 2001, www.w3.org/TR/wsdl.

Xu, Z, Martin, P, Powley, W & Zulkernine, F, 2007, Reputation-Enhanced QoS-based Web Services Discovery. ICWS pp.249-256.