International Journal of Software Engineering and Its Applications
Vol.8, No.2 (2014), pp.213-228
http://dx.doi.org/10.14257/ijseia.2014.8.2.21

Discovery and Visual Interactive WS Engine based on popularity: Architecture and Implementation

Selwa Elfirdoussi[1], Zahi Jarir[1] and Mohamed Quafafou[2]

[1]*LISI Laboratory, computer Science Department, Faculty of Sciences, Cadi Ayyad University, BP 2390, Marrakech, Morocco*
[2]*LSIS – UMR CNRS 6168, Domaine universitaire de St Jérôme, F-13397, Marseille Cedex 20, France*
[1]*s.selfirdoussi@ced.uca.ma , jarir@uca.ma, mohamed.quafafou@univmed.fr*

Abstract

Discovering a required web service still looks like finding a needle in a haystack despite enhanced proposed web service discovery methods. These methods which are based generaly on information retrieval techniques (word sense disambiguation, stemming, etc.), domain knowledge, ontology, etc. are more complex in practice. This motivates us to propose a seamless approach to discover a more appropriate web service metting user's requirements and also to invoke it using a generated GUI. To expriment our approach, we have developped a Discovery and Visual Interactive Web Service Engine based on popularity. The aims of DIVISE framework is to help users to select the right service according to their requirements and to provide them with an adequate GUI generated automatically to invoke the selected service (simple, composed or semantic). The proposed WS selection is based on popularity concept.

Keywords: *Web service Search; Web service discovery; Monitoring Web services; Tracking Web services, Invoking discovered Web Services, DIVISE Framework*

1. Introduction

Web Service Technologies are currently the most adopted alternative for implementing the Service-Oriented Computing paradigm, in which developers can discover and combine specific functionality offered by third-parties instead of developing them. However, Web services, published using registries, are still unfortunately not widely shared and reused as expected, despite the number of published services that increases more and more.

In fact, several approaches have been proposed in the literature to help users discovering a required web service. Among these approaches, we find for instance: (a) approaches based on semantic web enhancing web services' descriptions by annotating them using non-ambiguous concept definitions from shared ontology's [1], (b) approaches based on classic Information Retrieval techniques [2], such as word sense disambiguation and stemming for extracting pertinent information stored within WSDL documents, and so on.

Nevertheless, the problem is still representing a challenge. Most of the proposed approaches don't focus on discovering in all categories of web services (simple web services, composite web services, semantic web services, *etc.*). This is because each category has its precise structure (WSDL v1, WSDL v2, BPEL, OWL-S, etc.). Moreover, each approach extends UDDI structure by particular information dedicated to specific situation, which is not a standard or common approach. This let's UDDI not commonly used and is going out-of-date since it is ambitious to manage all kinds of Web services. Actually, researchers and

ISSN: 1738-9984 IJSEIA

practitioners have argued that UDDI has failed (or at least did not take off as anticipated), which is also shown in the shutdown of the most public UDDI registries (*e.g.*, Microsoft, IBM, SAP). Therefore, various less formal and more domain/usage-specific services have emerged [3].

In addition to these discussed challenges, the result returned is usually a textual data, which is not easily exploitable, and not in web form input to invoke directly the picked out web service. Furthermore, the selected web service must meet users' requirements and must be appropriate. Additionally, users need to have some information about reputation of returned web services such as frequency of use, duration of use, etc.

Our main contribution in this paper is to suggest a web search engine DIVISE (Discovery and Visual Interactive web Services Engine). This engine has the faculty to discover a required simple, composite or semantic web service and to help users select the appropriate Web service from a returned list. This list contains in addition to classical web service information a rate of its previous invocations, which is useful for deciding which one to select. Moreover, after selection, the returned result is a web form input to invoke directly a selected one from a proposed list.

The remainder of the paper is structured as follows: Section 2 presents a background of Semantic Web services and exposes our motivation. Section 3 and 4 summarizes some important related research studies of Web service discovery and Web service invocation. Section 5 details the module web service tracking to get the web service history uses. Section 6 describes our approach and its implementation in Section 7. Finally, our conclusion and some perspectives are given in Section 8.

2. Background and Motivation

Today, service-oriented computing represents the paradigm for the next generation of IT systems, with Web services as the base technology. Nevertheless, many challenges are still open and are mainly related to increase automation operations on web service such as discovering, composing web services, *etc*. In fact, we need fundamentally a seamless way to let computer programs implementing reliable, large-scale interoperation of web services. Among proposed solution is to create a Semantic Web of services whose properties, capabilities, interfaces, and effects are encoded in an unambiguous, machine-understandable form [4]. In this direction, Many researches were concentrated to adding semantic to web service for automate their invocation [3, 5, 6].

Recall that among mission of semantic web services is "to create infrastructure that combines Semantic Web and Web Services technologies to enable maximal automation and dynamism in all aspects of Web service provision and use, including (but not limited to) discovery, selection, composition, negotiation, invocation, monitoring and recovery". However, the potential of the Semantic Web will only become tangible only if it is widely used [7]. The pre-requisite therefore is that many people need to know what it is, and how it is used to determinate the eligibility of this web service, how to do it, and how to use it. In our sense, adding semantic information into web services by injecting human knowledge to automate operations like discovery, composition, etc. is a fastidious task.

In general, most semantic web and semantic web service technologies rely on or extend existing AI technologies [24] (*e.g.*, knowledge representation, planning, data integration, process algebras, *etc.*).

In this paper, we suggest a solution to help users to find a required web service meeting their preferences and also to execute it using a generated web interface. Our method consists on tracking all kind of services (simple, composite or semantic) and storing more important

information related to discovered, invoked or published web services in a database. This information will help us to find easily for example frequency of use of web services, their availability, last time they were used and during what period, the more reputation one, *etc*.

3. Related Work

We are considering not only the discovery of sample services, but also both composed and semantic ones. This section underlines a proposed related work by classifying them according to their contributions.

3.1. Discovery and Mediation: DIANE Service Description

The DIANE Service Description called DSD is a framework that can discover and invoke web services automatically [8]. DIANE middleware is defined to automatically respond to a client request by making connections with various web services. So the main feature of DSD is the discovery of web services and the cooperation between systems to automatically compose them and execute the resulted composite service to deliver the results to the applicant. DSD does not support the automated invocation with the generic input and output parameter and doesn't take into account the quality of the web service in the selection module.

3.2. Filtering and Selecting Semantic Web Services Composition with Interactive Techniques

To demonstrate the utility of semantic Web service descriptions for service composition, a team from the University of Maryland has created the interactive composition approach using matching algorithms whose goal is to help users to filter and select Web services during the creation and composition [9]. Indeed, this team has implemented a prototype system that can compose web services published and provide filtering capabilities to deal with the case where a large number of similar services could be available. Nevertheless, filtering and selecting web service is based on its semantic description that can be controlled by the service provider and not by its popularity. Also the system cannot invoke the selected service and run it. It requires more effort and time for the client Interactive discovery and composition to handle complex web service.

3.3. Easy web service discovery: A query-by-example Approach

Web services have gained tremendous popularity among software developers. This popularity has motivated developers to publish many descriptions of Web services in UDDI registries. In [11], authors have presented a new method to discover Web services called WSQBE, which aims to facilitate the application and assist clients in returning a short and specific list of candidate services. The WSQBE discovery process is based on a mechanism that reduces the search space automatically and makes this approach more effective. The approach is very convenient for frequent users, except that it does not show the reliability of the service.

3.4. Automated Generation of Composite Web Services based on Functional Semantics

Most studies on the generation of web services create composite services by chaining available inputs and outputs, but do not consider their functional semantics [12]. Therefore, the results can be unsatisfying and against users' intentions. To solve these problems, authors of [13] proposed a method of composition that explicitly specifies and uses functional

semantics of Web services. More precisely, the proposed method is based on a graph model, which represents the semantic description of Web services. In this method, the web service's specifications are stored in the proposed graph model. A composite service is automatically built by searching the graph based on the user's requirements, and available services are dynamically classified into simple services, which provide the required functionality. The proposed method improves the accuracy of the generated composite services, and can bring together two cases where both services have the same description of the method proposed specification, but may work differently which can be a source of error.

3.5. Dynamic Web Service Discovery Architecture based on Overlay Networks

Discovery mechanisms of WS are critical to the overall utility services. Until now, discovery mechanisms based on the UDDI standard rely on centralized directories and many specific areas, which poses problems of stress information such as performance bottlenecks and fault tolerance. In this context, decentralized approaches based on peer to peer overlay networks have been proposed by many researchers as a solution. In [14], authors propose a new P2P overlay network infrastructure designed for Web Services Discovery. The structure of Web service discovery on a P2P network must determine the node that stores the Web service element that satisfies a criteria range. The work done is to present and analyze Nippers. So, the approach supposes that the node discovered is correct and gives the right response according to the user's request. In second hand, the proposed work doesn't take the popularity of the selected web service and its exploitation by an automatic invocation or the semantic interpretation.

4. Related Work on WS-Invocation

The second more important challenge faced by web services technology is an automating invocation of discovered web service by offering an appropriate user interface. So in this section we try to expose some related work proposing some solution. After that we give our critics.

The work described in [15] proposes a framework for a client to dynamically invoke web services. The framework can increase the use and reliability of web services invocation in a dynamic and heterogeneous environment. This work designs and realizes a dynamic Web Service framework, which enhances the reliability, transparency and dynamics of Web Service invocation.

WSIF is Web Service Invocation Framework [16] gives to its users a uniform API to access WSDL-described Web Services. To become a dynamic provider, it is necessary to implement "WSIFDynamicProvider" interface. The interface is used by WSIF runtime to convert WSDL port into a "WSIFPort", which is used to execute operations. Such design enables protocol bindings to be swapped dynamically and even allows use of hand-coded port implementations in conjunction with pre-existing port implementations seamlessly. The WSIF user does not need to be concerned about how WSIF invocations are implemented but if necessary the user can override default behaviors allowing for exile execution patterns.

DAIOS [17] is a message-based service framework that supports implementation of SOAs, enabling dynamic invocation of SOAP/WSDL-based and "RESTful" services. The framework internally splits up into three functional components: the general DAIOS classes representing the core of the framework have the role of orchestrating other components, the interface parsing component which is responsible for preprocessing (binding) and the invoker component which conducts the actual Web service invocations using a REST or SOAP stack.

Clients communicate with the framework front-end using DAIOS messages which are DAIOS' internal data representation format.

All these presented approaches propose an XML interface [18], which means that the user must have some capabilities to manipulate the framework. In addition the proposed interface does not take into account the dynamic generation for the input and output parameters. Moreover some of them propose just a dynamic invocation for a simple web service and doesn't consider the other categories like semantic WS or the composite WS.

5. Web Service Tracking

Discovering a more interesting web services according to users' requirements is a challenging task because of their diversity (simple, composite and semantic), federation and also dissimilarity of web services registry architecture. Motivations for tracking web services arise from the perspective of both users and service providers.

Users are interested in understanding which web service among returned list is more used, more available, etc. to decide which one is appropriate for their requests. Also web service providers that are using other services to provide composite web services would like to know how a specific and/or composed web services are behaving.

In fact, web services tracking approach is in our opinion a very useful approach to meet both users and service providers' requirements. Monitoring discovery, invocation and publishing users' requests is very helpful in analyzing availability, frequency use, quality, etc. of web services.

By studying monitoring results of users' behaviour, discovering or composing web services operation would be able to optimize itself by selecting a more appropriate web services to improve performance. This approach does not require any additional information to be injected into web services structures or web services registries architecture to perform such tracking.

To achieve our goal, we need first to (a) determinate which information must be captured? (b) how to capture and store this information? (c) how to use it to meet users or service providers' requirements.

Consequently, we proposed an algorithm to track each used Web Service. The implementation of this algorithm is exposed in more details in subsection 7.1. The code is presented as follow:

```
// IdentifiedOperationType = Invoke Or Inquiry
Procedure trackingWS (WS, IdentifiedOperationType) return idWS
  WSTrackingDate = getDate();
  clientIP = getUsedIPAddress();
  idWS = getAccessPoint();
  timeOfUseWS = getTimeOfUseWS();
  If (IdentifiedOperationType = "inquiry")
     Query = used query to inquiry WS
     Insert in DBLog.Profile(WSTrackingDate, clientID, idWS,
query, "Inquiry", timeOfUseWS);
  Else if (IdentifiedOperationType = "Invoke")
     Insert in DBLog.Profile(WSTrackingDate, clientID, idWS, "Invoke", timeOfUseWS);
  End If.
  ParseWS (idWS); // parsing WSDL, or BPEL or OWL-S
  Insert in DBLog.WServiceDetails (idWS);
End Procedure
```

6. Interactive WS Engine DIVISE Architecture

6.1. Approach Description

Recall that a main goal of our approach for WS discovery is to help users to find web services using WS popularity gained from tracking users' behaviors and decisions. This approach takes also in consideration users profile to better meet WS selection using automate approach or interactive approach (*i.e.*, semi-automatic selection).

Among the highlights of this approach, we cite for instance:

- Diversity of tracking WS (simple, composite and semantic WS),

- WS Popularity summarizing frequency use, reputation, availability, failure frequency, *etc.*, of previous used WS.

In fact, we suggest a solution Interactive WS engine DIVISE which is based fundamentally on web services popularity obtained from stacked and stored information of used WS. The main components involved by our architecture are WS-Discovery module, WS-extracting module, WS-Invoke module, WS-Monitoring module and WS-Parser module as depicted in Figure 1. Each of these modules will be described in details in next subsections.

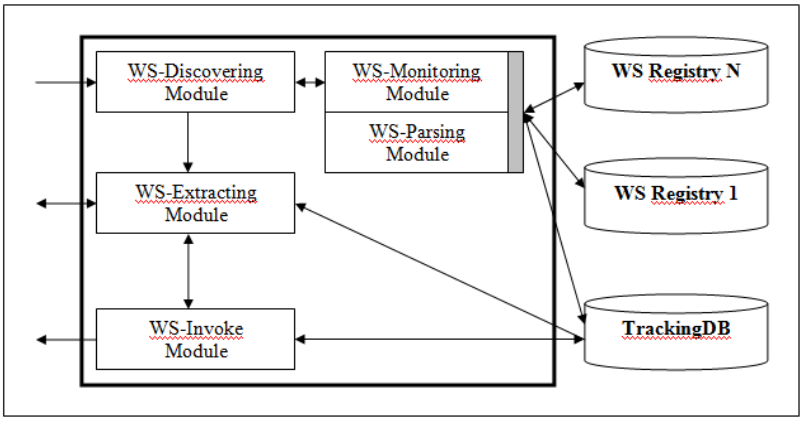

Figure 1. Interactive WS engine DIVISE Architecture

6.2. WS-Discovery Module

Selection of partner services that fulfill the functional requirements of users is the first step to realize an interactive discovery. Users have to specify a query (keywords or a composition of words) that describes web services to find. This query will be addressed firstly to WS-Discovery module for discovering all kind of WS (simple, composite or semantic) according to users' preferences (specific domain for instance). This is done by interactive with most known WS Registries and also with our proper Tracking Database.

6.3. WS-Extracting Module

When discovery process is terminated, WS-Extracting Module will return a list of discovered WS. To get more information about WS use, it will consult Tracking Database. A returned result list containing discovered web services is structured as follow:

Name of service – access point or URL – type – last invoked date and time – average of used time – frequency of use. This method allows users to have an idea about popularity of web services and to estimate if it's appropriate.

This information will help users select the appropriate service. Moreover the process of discovering could be automated just by adopting some rules in user's profile. The best way to know if a web service is correct, proved, available, and more popular; we suggest a link in returned page showing graphic presentation of historic use of a selected web service (number of time and date, users IP address, *etc.*).

6.4. WS Invoke Module

Since the user is convinced that he has found a required web service. A WS Invoke method is fired to generate a web interface related to web service (simple, composite or semantic) to invoke. This interface takes in consideration validity of required type of input parameters using some classes named Validators.

6.5. WS-Monitoring Module

The role of this module is to track exchanged messages between users and WS providers when discovering, invoking or publishing WS. It captures in addition to URL or access point of concerned WS others pertinent information such as date and time of use, period of use, *etc*.

To have more details about a WS, it communicates access point or URL to WS-Parser.

6.6. WS-Parser Module

The role of this module is to parse a WS description to get all required information using an URL or a specific access point. More precisely, this module contains a specific parser to handle each type of web services; for instance WSDL parser, BPEL parser, *etc*.

7. Implementation of DIVISE Architecture

The main objective of this section is to expose the implementation of principal modules of DIVISE Architecture.

7.1. WS-Monitoring Module Implementation

Among objective of this module is to trace all exchanged message between users' queries and WS registries when discovery web service is called. To more explain the behaviors of such module, we take as example UDDI registry and SOAP protocol. In this case, we have used JUDDI Log which is an API developed on JUDDI to retrieve and display the SOAP message (request & response) exchanged. Once a result is found by WS registry, this module stores a communicated user's query (keywords, etc.) and also communicate information for instance access point related or URL of returned WS. This information is obtained by parsing SOAP messages.

Moreover parsing WS URL gained more information about discovered WS, which is the role of WS-Parsing Module. Three categories are implemented: WSDL parser, BPEL parser and OWL-S parser. Once the needed information is collected, it will be stored in specific tables in our Tracking database.

To implement this module, we have changed JUDDI API in a generic way to monitor UDDI registries. The used code modification can seamlessly be used for any kind of WS registry. In fact, we have added a listener in the "Registry Engine" which represents a class

that captures both incoming and outcoming SOAPMessages. A fragment of code is presented as follow:

```
SOAPBodyElement soapRespBody = new SOAPBodyElement(responseElt);
String responseMsg = soapRespBody.toSTring().trim();
com.ucam.juddi.ParseResponseMsg.execute(responseMs);
```

The developed method execute of "com.ucam.juddi.ParseResponseMsg" class consists in parsing SOAP response and writing in the log file (JUDDI) the value of each XML tags of corresponding SOAP message. Thereafter this module is able to collect all WS information by parsing SOAP response using for instance DOM Parser or another API.

In case of publish and inquiry request, DIVISE framework tries to get WS access point in order to extract needed information. From this access point WS-Parser Module parses and collects information that will be thereafter stored in tracking database.

Our objective, for publish or inquiry request, is to have address point for web services for extracting information. This address will be parsed by WS-Parser Module and collected information will be thereafter inserted in tracking database.

In addition to listing publish and inquiry request, WS-Monitoring has also to trace SOAP messages for invoking a web service.

In fact, we have used the approach Handler proposed in Axis framework to collect information during an invocation of the web service. In this case, the more trivial solution is to use log file of Axis to collect exchanged message. However, this solution needs to add a specific listener or a trigger program on log file in Axis to know if an exchange is handled, which presents some limitations.

Therefore, we have used our developed class "PerformanceHandler" that inherits from "BasicHandler" class of Axis instead of "Handler" class proposed by Axis. In addition, we have set service configuration in order to invoke our class at every web service call. A fragment of code is presented as follow:

```
<ns1:responseFlow>
   <ns1:handler type="java:org.apache.axis.handlers.SOAPMonitorHandler"/>
    <ns1:handler type="log"/>
</ns1:responseFlow>
<ns1:handler name="log" type="java:com.doctorat.soap.monitor.PerformanceHandler">
  <ns1:parameter name="LogHandler.fileName" value="axis.log"/>
</ns1:handler>
```

Furthermore, we have developed a specific method called "invoke" within PerformanceHandler that consists in retrieving In-Message and Out-Message of exchanged SOAP messages, parsing all SOAP message, collecting and storing pertinent information of invoked WS.

7.2. WS-Parser Module Implementation

When a WS is discovered or invoked, WS-Parser module is fired to find a location throws communicated access point or URL for parsing its description in order to retrieve and use its contents. Since structure of simple, composite and semantic WS is different, we have created for each category a corresponding class. Attributes of these classes are mapped from tags of corresponding WS. In each of these classes, a generic and static method named "load" is defined. This method reads WS file from access point or URL and affects captured values

into attributes of the concerned class. WS file might be described in WSDL language (v1 or v2), BPEL language or OWL-S language.

To parse those files, we have used WSDL4J for WSDL description, verbose for BPEL description and the OWLS API for OWL-S description. Below, a fragment of WSDLParser code:

```
public class WSDLParser extends WSDLReaderImpl{
        String wsdlUri ;
        Definition wsdlDef;
        String name;
        String serviceName;
        String location;
        Map services;
        Map ports;
        Map binding;
        Map imports;
        Map messages;
        Map namspaces;
        String targetNameSpace;
```

7.3. Tracking Database Schema

To better exploit monitoring of discovered and invoked web services, we have designed a relational schema. This schema contains the most important attributes collected from parsing process of concerned WS in addition to those related to users' queries. Recall that in a previous section we have shown how we capture the access point or URL of discovered or invocated WS. In this section we will focus on how to collect pertinent WS information from this access point?

The defined database schema contains a main table named "log" which has a role to persist basic information related to user's navigation such as URL or access point of WS, discovery or inquiry time and date, Client IP, Provider IP, etc. In addition, it contains according to simple, composite and semantic WS one table WS service which contains a type field indicating a type of web service. This table is mapped from defined parser classes.

```
public static void saveWSDL(String fileUrl,String idParent){
WSDLParser wsdlFile = new WSDLParser(fileUrl);
//Call save log
String idws = saveService(wsdlFile,wsdlFile.getLocation(),"INVOKE",idParent);
//Save Operations and parameters
saveOperations(idws,wsdlFile);
```

For each kind of web service, we define in our system a class responsible to insert the usual information in tables. These classes are named WSDLInsert, BPELInsert and OWLSInsert.

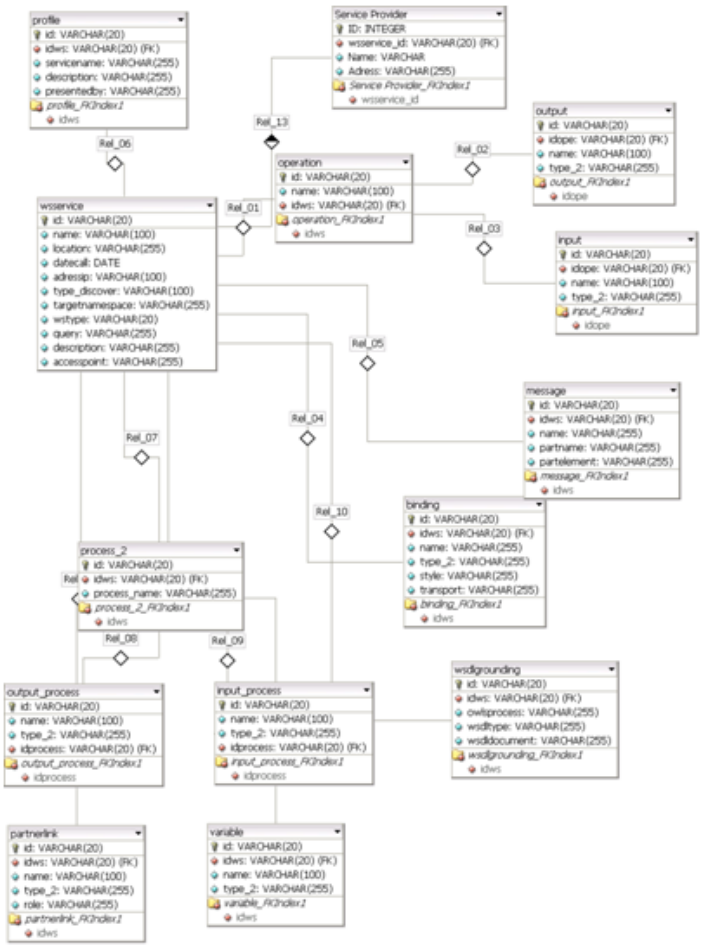

Figure 2. Database Schema

7.4. WS-Invoke Module Implementation

In our framework, we choose to call web service using its location in the Invoke process; this issue get process independent and can be called from the discover process or from itself. As described in the previous section, the process supports 3 types of web services (WSDL, BPEL and OWL-S). Bellow the definition of the code that we execute to invoke WSDL and BPEL web services. It's necessary to note that BPEL process is a web service that is simple to use.

So far we have described the method invocation of web services for the three kinds. The "WSInvoke" Process allows the users to read WS File description and displays parameters in the invocation form. The process doesn't require any special knowledge of web services technologies; also, the user doesn't need to be familiar with the Framework APIs to get the process, create the messages, and execute the operations.

For a given WS document created by WSDL, BPEL or OWL-S, WSInvoke read the Web service implementation file from the service provider and parses it for various information. In this way we have enough information to encode a SOAP request to the Web service implementation using Axis. In the client code this is accomplished using forms called "list_parameter.jsp" displaying the list of operation to invoke and the input tag for each operation parameter. To parse a web service document, we have used wsdl4j [19] for WSDL

document, WS-BPEL [23] for process BPEL and OWL-S API for the semantic services [21].These APIs allows the user to get all necessary information from the file description. Then we will use Axis "org.apache.axis.client.Call" [20] to actually make the SOAP request to the server and wait for the response.

DIVISE includes a stub compiler (called Tag compiler) that takes as input WS File and generates JavaBeans to represent schema types defined in WS file as described in the section "Web Services Format", Java interface that corresponds to tag and stub that is providing uniform access to web service hiding underlying protocol bindings. However, the generated code is not completely static; it use dynamic providers to actually execute the operations so it is possible to replace DIVISE protocol binding implementations used by the stub.

In this part, we define an "ExecEngine" to execute an OWL-S service using the "OWLSFactory" class defined is the OWL-S API [21], as described in the "web service format" section process is a definition of web service that pointed to grounding [22]. This tag is used to give information about the web services access point and operation with their inputs and outputs.

The most interesting feature of our framework is that each process can be executed separately. Therefore, the application is accessible for exploiting web service using their file description or from the discover process. In the next section, according to each type of web services we give an illustrated example for the invocation module.

8. Experiments

A study of different web services has been used to evaluate our proposed approach and architecture. In this case, we present a scenario when users use interactive WS engine DIVISE to find and invoke web services. An example of such inquiry and also a returned list is shown in the next paragraph.

The Figures 3 and 4 mentions how to discover the web services using a key and the result of our query.

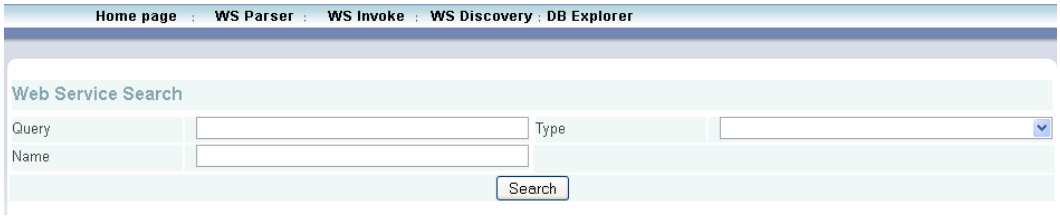

Figure 3. The Discovery Search

This figure presents in the first scream the form to find a web service. The search form contains three criteria:

- Query: the user's query.
- Type: the web service type (simple, composite or semantic).
- Name: the entire or part of the web service name.

Figure 4. The Discovery Result

The user's can start the web service by using or not the criteria. The list of result contains all useful information of the web services as the name of web service, the web service type, the query used for selection, the last use date and the access point to web service.

Otherwise, we have the possibility to call the WSInvoke from the "execute icon" or directly from the menu "WSInvoke" selecting the type of WS and the location of its file description. After calling the button invoke, a "list_parameter.jsp" display the list of the service operation, and for each one, the input value and format. In this example, the input of the operation "getFullName" is a complexType described by the Tag <complexType> as flow:

```
<complexType name="Person">
<sequence>
   <element name="birthday" type="xsd:dateTime"/>
   <element name="firstname" type="xsd:string"/>
   <element name="name" type="xsd:string"/>
   <element name="nbChild" type="xsd:int"/>
   <element name="perCode" type="xsd:string"/>
  </sequence>
 </complexType>
```

In the Figure 5, we present the dynamic invocation for the three type of web service. The "WS invoke" menu contains three links (WSDL Invoke, BPEL Invoke and OWL-S Invoke). The scream for calling WS invoke is the same. So, for invocation, user must enter the access point of the web service he wants to invoke according to the type of web service (WSDL for simple, BPEL for composite and OWL-S for semantic).

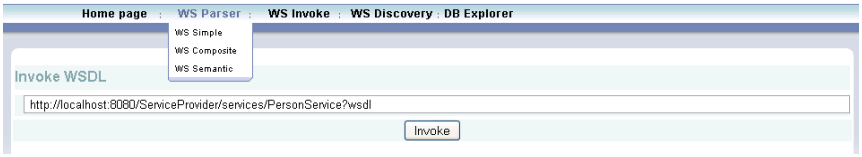

Figure 5. The Form to Call the Invocation of Web Service

After selection the web service access point, the user chooses "invoke" to get the invocation form described in the Figure 6.

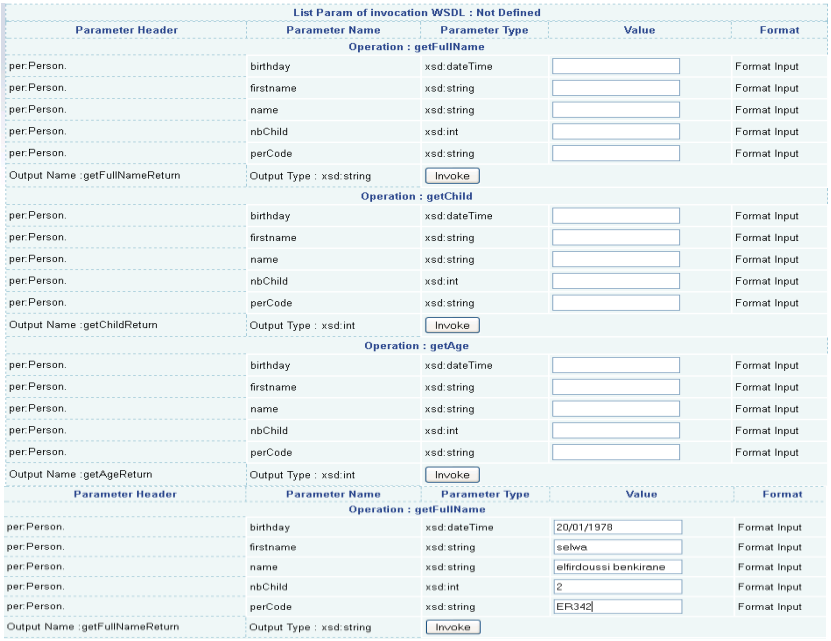

Figure 6. The Invocation Form for Web Service Simple

The Figure 6 presents for each operation defined in the WSDL File the parameter list to invoke it and for each parameter we expose his type (string, datetime, *etc.*). Each operation can be executed by using the bottom "Invoke". The result of invocation is the response for the SOAP Request that we constructed from this forms and we passed to the call Axis class as described in the previous section.

Figure 7. The Result Invocation Scream

For the BPEL process, there are two ways to invoke using our DIVISE. The first method is to invoke the web services generated from the BPEL process, which is similar to WSDL Invoke explained in the previous paragraph. The second way consists in parsing the BPEL Process which defines the sequence of Activities mentioned in the process and executes it in Ajax. We choose Ajax to call each activity because in the BPEL definition we can have an asynchrony process that interacts with clients. We recall that Ajax is an API that executes request without full page refresh.

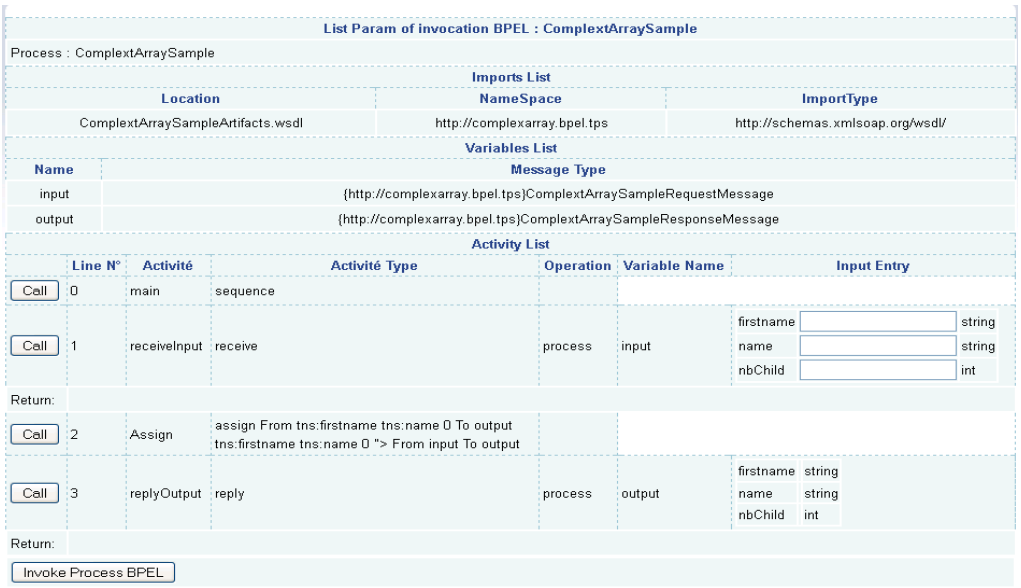

Figure 8. The Dynamic Invocation form for Web Service Composite

The Figure 8 presents the activities contained in the web service composite and for each activity the list of their parameters. In our example, the composite web service is composed using the activity main, receiveInput, assign and replyOutput. The second activity contains a parameter as a complex type that we import from the WSDL location. The button "Invoke Process BPEL" is the workflow to execute the succession of activities by order.

Figure 9. The Result of the Invocation Form for Web Service Composite

For the OWL-S type of web service, the invocation process is using the grounding file. So, the web service "BabelFishTranslator.owl" is composed from three parameters (input language, output language and the input string). The aims of the web service as described in his name are to translate string from language to another one.

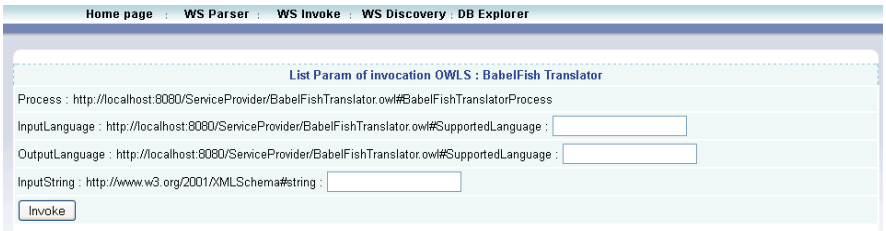

Figure 10. The Dynamic Invocation Form for Web Service Semantic

The Figure 11 presents the result of invocation the translation of "Hello World" from English to French.

Figure 11. The Result of the Invocation of Web Service Semantic

9. Conclusion

Our main contribution deals with automatic or semi-automatic web service discovery taking in consideration users preferences. More precisely, we propose an approach based on tracking all kinds of web services discovered and/or invoked by users. In this fact, we have implemented a related architecture that has the advantage to monitor exchanged SOAP messages, parse discovered or invoked WS and store captured pertinent information to enrich our database. A presented solution, named DIVISE (Discovery and Visual Interactive WS Engine), takes advantage of knowledge stored in this database to help users to select more appropriate WS regarding to their needs, or service providers to build more complex one by composing selected WS. This selection is based on WS popularity, frequency of use, availability, *etc*. An experiment of DIVISE is also presented.

Finally, we plan to more evaluate scalability of proposed approach and Web search engine DIVISE using different WS registries conform or not to UDDI structure.

References

[1] S. A. Mcilraith and T. Cao Son, H. Zeng, IEEE Expert / IEEE Intelligent Systems - EXPERT, 46, 53, **(2001)**.
[2] N. Kokash, Proceedings of the Third Starting AI Researchers' Symposium, **(2006)** June, Trentino, Italy.
[3] J. Wang and J. Zhang, P.C.K. Hung, Z. Li, J. Liu, and K. He, Proceedings of 13th IEEE International Conference on High Performance Computing & Communication, **(2011)** September 2-4, Banff, Alberta, Canada.
[4] A. Ankolekar, M. Burstein, J. R. Hobbs, O. Lassila, D. L. Martin, S. A. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara and H. Zeng, "Proceedings of the First International Semantic Web Working Symposium (SWWS): Infrastructure and Applications for the Semantic Web", Stanford University, California, USA, **(2001)** July 30-August 1.
[5] K. Srivastava and S. Kumar Malik, IEEE Expert / IEEE Intelligent Systems - EXPERT, vol. 24, no. 31, **(2000)**.
[6] M. Klein, B. Konig-Ries and M. Mussig, International Journal of Web and Grid Services - IJWGS, **(2005)**, pp. 328-364.
[7] M. Stollberg, M. Moran, L. Cabral, B. Norton and J. Domingue, "Semantic Web Education and Training Workshop", Beijing, China, **(2006)** September 4.
[8] U. KÄuster, M. Klei and B. KÄonig-Ries, Second Semantic Web Service Challenge Workshop, Budva, Montenegro, **(2006)**.
[9] E. Sirin, B. Parsia and J. Hendler, IEEE Expert / IEEE Intelligent Systems - EXPERT, vol. 42, no. 49, **(2004)**.
[10] S. Stupnikov, L. Kalinichenko and S. Bressan, "Advances in Databases and Information Systems", vol. 216, no. 231, **(2006)**.
[11] M. Crasso, A. Zunino and M. Campo, "Science of Computer Programming - SCP", vol. 144, no. 164, **(2008)**.
[12] M. Vukovi, "Rapid Integration of Software Engineering Techniques", no. 129, no. 144, **(2005)**.
[13] D. Shina, K.-H. Leea and T. Suda, Journal of Web Semantics, vol. 332, no. 343, **(2009)**.
[14] S. Sioutas, E. Sakkopoulos, C. Makris, B. Vassiliadis, A. Tsakalidis and P. Triantafillou, Journal of Systems and Software, vol. 809, no. 824, **(2009)**.
[15] G. M. Tere, B. Jadahav and R. Mudholkar, Advances in Computational Research, vol. 78, no. 82, **(2012)**.
[16] J. M. Duftler, K. Nirma. Mukhi, A. Slominski and S. Weerawarana, IBM T.J. Watson Research Center, **(2001)**.
[17] P. Leitner, F. Rosenberg and S. Dustdar, IEEE Internet Computing, vol. 72, no. 80, **(2008)**.
[18] T. Erl, Prentice Hall, Upper Saddle River, NJ, USA, **(2004)**.
[19] L. Hang, Watson Research Center, USA.
[20] R. Whitney, Open Content (http://www.opencontent.org/opl.shtml), **(2010)** May 4.
[21] E. Sirin and B. Parsia, "The OWL-S Java API", Proceeding in third International Semantic Web Conference, San Diego, California, USA, **(2004)** July 6-9.

[22] M. Klein, B. Kong-Ries and M. Mussing, International Journal of Web and Grid Services, vol. 328, no. 364, **(2005)**.
[23] A. Iyengar, V. Jessani and M. Chilanti, IBM Press, **(2007)**.
[24] S. J. Cha and K. C. Lee, International Journal of Software Engineering and Its Applications, vol. 177, no. 188, **(2013)**.

Authors

Selwa Elfirdoussi, has obtained a diploma of Engineer in Software Engineering from ENSIAS School of engineering, Mohamed V Souissi University, Rabat, Morocco in 2000. Actually, she's a PhD student at Faculty of sciences, Cadi Ayyad University in Marrakech, Morocco since 2008. Her research interest is focalized on service-oriented computing and Web service technologies.

Zahi Jarir, received his postgraduate degree in computer science in 1997 on Natural Language Processing at Faculty of Sciences in Rabat, Morocco. From 1997 to 2006, he was assistant professor at Faculty of sciences, Cadi Ayyad University in Marrakech, Morocco. In 2006, he received academic accreditation from Cadi Ayyad University. Currently, he is a professor of Computer Science at Faculty of Sciences of Cadi Ayyad University. His research interests at LISI laboratory lie mainly with the field of Service-oriented computing and technologies, Cloud computing and security, Computational reflection and Meta level architectures, Adaptive and Mobile Middleware, and Customization techniques of Web Services and Applications.

Mohamed Quafafou, did his PhD Thesis in 1992 on Intelligent Tutoring Systems at INSA de Lyon, France. From 1992 to 1994, he was ATER at INSA de Lyon and then at Nantes Faculty of Sciences. From 1995 to 2001, he was assistant professor at the Nantes University. During that period, he developed research on Rough Set Theory, concepts approximation, data mining, web information extraction and participated actively with France Telecom to design a new web system dedicated to French web analysis for discovering emergent web communities. He was also chief-scientist at GEOBS where he headed the Geobs Data Analyzer project, which was developing a spatial data mining systems with application to environment, marketing, social analysis, etc. From September 2002, he was professor at the Avignon University and moved in 2005 to the Aix-Marseille University where he joined the LSIS CNRS and leads research on Data Mining Theory and Applications focusing on new contexts for learning (crowdsourcing, interconnected data, and big data) with application to user's behavior analysis, web services, cloud automatic auto-scaling, social network analysis, etc.

