Modern Applied Science; Vol. 6, No. 9; 2012
ISSN 1913-1844 E-ISSN 1913-1852
Published by Canadian Center of Science and Education

WSDATool: A Novel Web Service Developer Assistance Tool Using a New Complementary Service Publishing Method

Fardin Abdali Mohammadi[1], Naser Nematbakhsh[1] & Mohammad Ali Nematbakhsh[1]

[1] Department of Computer Engineering, Faculty of Engineering, University of Isfahan, Isfahan, Iran

Correspondence: Fardin Abdali Mohammadi, Department of Computer Engineering, University of Isfahan, HezarJarib Street, 81746-73441, Isfahan, Iran. Tel: 98-912-605-2405. E-mail: ebdalimo@ce.sharif.edu

Received: July 27, 2012 Accepted: August 20, 2012 Online Published: August 24, 2012

doi:10.5539/mas.v6n9p42 URL: http://dx.doi.org/10.5539/mas.v6n9p42

Abstract

Web service technology (WST) is a service-oriented architecture implementation framework that makes designing component-based internet applications possible. At present, many providers offer their services as web services. Current WST suffers from the lack of an integrated tool to assist web service developers. In WST, the services are published publicly, and their descriptions are stored in service directories. These descriptions contain valuable information about the work of different software teams throughout the world. However, with the increasing number of web services, searching for services is difficult and time-consuming. Furthermore, in current service directories, there is a little knowledge about the services, and extraction of useful information to be utilised by developers is not easy. In this paper, in order to increase the knowledge of what is available in service directories, a structure is presented by interlinking WST entities by using some defined semantic relations. The proposed structure provides a framework and a tool named WSDATool to develop new web services using information from published services or to refine current published web service descriptions. In experiments, services designed with the assistance of the WSDATool are more coherent and well designed.

Keywords: developer assistant, semantic relation, software development, web service

1. Introduction

Software systems are currently very dependent on software components. Service-oriented architecture is one of the best practical architectures for component-based systems. In this architecture, service providers present some services to be utilised by users and applications. Web service technology (WST) is one of the solutions provided by this architecture. A web service is a software component that presents an API accessible via the web.

Hoffmann et al. (2007) by investigating 15 million programmer queries from the MSN search engine, showed that 34.2 percent of programmers' queries involved finding suitable application program interfaces (APIs). By the same way, web service developers search the Internet to find the information they need. Anyway, one of the main problems with developing web services is the lack of an assistance tool to help developers create coherent and easy-to-discover web services.

Furthermore, most service descriptions are provided by integrated development environment IDE tools automatically. The generated service description are full of anti-patterns and bad coding practices (Crasso, Mateos, Zunino, & Campo, 2010; Crasso, Rodriguez, Zunino, & Campo, 2010; Juan Manuel, Marco, Alejandro, & Marcelo, 2010; Juan Manuel Rodriguez, Zunino, & Campo, 2010; Lu et al., 2010; Mateos, Crasso, Zunino, & Campo, 2010; Rodriguez, Crasso, Zunino, & Campo, 2009, 2010; Sneed, 2010) that reduce the chance of a service being discovered correctly (Juan Manuel et al., 2010). This situation motivated the current study in which a tool for overcoming this problem is proposed. The idea is to utilise current published web services to assist web service developers. The published services stored in service directories contain valuable information that can be used to help developers develop more powerful and coherent web services. This information is the result of work done by different software teams throughout the world.

However, extracting useful information from service directories is a difficult and time-consuming process. With the increasing number of web services, searching for services is difficult and time-consuming. For example, at the time this paper was being written, *Seekda* (a web service crawler available at http://webservices.seekda.com/) had indexed 28,606 working web services from 7,739 providers. Furthermore, service directories only offer the

simple similarity searching operation. Even the enhanced similarity matching operation is very time-consuming, and in the best situation, it returns services that are similar to the user's request.

To assist developers, we need to extract more information from service directories and organise them in a knowledge database. By querying these knowledge based service directories, developers can extract required information. Extraction of information cannot be accomplished using the simple searching APIs provided by traditional service directories or enhanced similarity matching operations. To date, efforts to enhance service directory search operations have mainly concentrated on the precision of the similarity matching operation. Examples include semantic descriptions of web services (Iqbal, Sbodio, Peristeras, & Giuliani, 2008), similarity matching algorithms based on semantic descriptions (Plebani & Pernici, 2009), and similarity calculation algorithms based on ontology (Klusch & Kapahnke, 2009). These efforts have not increased the knowledge about the contents of service directories and cannot assist web service developers.

In this paper, a method of publishing web services by interlinking web service entities based on the concept of semantic relations is introduced. Some semantic relations are defined and an interlinked graph of web service entities is constructed. A software tool is developed on top of this structure to assist web service developers and named WSDATool. Services designed with the assistance of the proposed system are more coherent, and free from anti-patterns. WSDATool can also be used to refine current service descriptions.

The main contributions of our work are as follows:

- We present a structure for increasing the knowledge about the contents of service directories. In this structure, the WST entities are interlinked based on semantic relations that may exist between them. This part is an extension to our previous work in (Fardin, Naser, & Ali, 2012).
- We present a method to extract knowledge from interlinked structure and an expert agent to assist developers using tis knowledge.
- We present a novel tool to assist web service developers based on the presented structure.

The presented system is capable of presenting usable information at a lower cost to system developers, and it helps them develop web services. By lower cost, we mean that the speed of performing high-level queries can be increased by using the presented structure and that these queries are described by a user-friendly language. In this paper, real services were used for expressing the definitions and examples.

1.2 Related Work

Several researches (Crasso, Mateos, et al., 2010; Crasso, Rodriguez, et al., 2010; Juan Manuel Rodriguez et al., 2010; Lu et al., 2010; Mateos et al., 2010; Rodriguez et al., 2009, 2010; Sneed, 2010) are investigated the problems of current registered web service descriptions and the effect of these problems on service discovery and usability. The results of these researches are mainly list of anti-patterns that should be removed in order to improve service usability.

Several tools have been developed to assist in the annotation of web services. Dietze et al. (Dietze, Yu, Pedrinaci, Liu, & Domingue, 2011) created an editor to edit and search for linked-data based services. Linked-data based services are a method of describing web services by connection service descriptions to linked-data (Bizer, Heath, & Berners-Lee, 2009) cloud entities. Maleshkova et al. (2009) proposed an editor named SWEET to annotate semantic web services. This editor is useful for annotating web services, and it creates an HTML description of the service.

Birukou et al. (2007) presented a method to assist system developers. In this method, for each user request, the invocation and performance of relevant services are monitored. By monitoring the performance of services, QoS can be achieved in the proposed system. At first, operational data are modelled. Then, by using the semantic similarity between the user request and the registered activities, the system makes recommendations.

Recommender systems have been attracting the attention of software researchers. Several methods based on recommender systems are presented for extracting information and for helping service developers. Robillard et al. (2010) provide an overview of available recommender systems that assist application developers. These methods can be divided into the following two types:

1) Recommender systems (Bhaskar, Claudia, Avare, Claudio, & J., 2004; Birukou et al., 2007; Blake & Nowlan, 2007; Ichii, Hayase, Yokomori, Yamamoto, & Inoue, 2009; Sellami, Tata, Maamar, & Defude, 2009; Shripad & V., 2005; Zheng, Ma, Lyu, & King, 2009): These systems try to discover the most likely web services based on the behaviour of other users and by preserving the system history.
2) Search engines (Dong, Halevy, Madhavan, Nemes, & Zhang, 2004; Vu, Hauswirth, Porto, & Aberer,

2006): These systems try to discover the best web service candidate based on the similarity of a user query to registered web services and the aid of a ranking algorithm.

The output of both methods is the most similar service corresponding to the user request. Although these methods were designed to assist service consumers, they can also be used by service developers.

These systems can be used by web service developers to create better services. Ichii et al. (2009) present a recommender system based on collaborative filtering to propose software components. However, Zheng et al. (2009) propose a collaborative filtering system for web service recommendations based on QoS. Both presented methods in (Ichii et al., 2009; Zheng et al., 2009) utilise the history of user activity and present a method for expressing the similarity between users. Blake and Nowlan (2007) present a method for web service discovery and candidate recommendation based on syntactical matching. For this purpose, search algorithms, service ranking and syntactical similarity matching are used.

Fardin et al. (2012) construct an interlinked graph of WST entities named Semantic Interlinked Graph in the service directories in order to augment service directories with knowledge. In this graph, entities are interlinked using some sort of semantic relationships that may exist between them. They shows how these relations can be formally defined using ontology, implemented using RDF, and extracted using the SPARQL language. Furthermore, they show how some limitations of the current service directories such as service versioning and registering composition plans can be resolved.

Calculation of similarity between WST entities are one of the important task in service discovery operations. Plebani and Pernici (2009) presented a method to measure the similarity between two web services. This method is used for the replacement of services in case of failure. They propose a formula in which the similarities between web service operations are calculated based on the similarity between arguments name and their data types. Also they extend their method calculate the similarity between semantic web services too. Crasso et al. (2008) present a method to calculate similarity of user query and web services as the angle between two vectors of words.

2. Structures, Methods and Tool

The core element of this work is a Semantic interlinked graph *(SIG)* of WST entities. In this structure, a graph of web service entities is constructed based on some semantic relation between them. *SIG* is developed in previous work in (Fardin et al., 2012) and we enhance it in this paper make it usable for developers assistance tool. This structure is a complementary facility for service directories that increases the knowledge about the contents of service directories. By using this structure, high-level queries are possible. The following section presents the construction of *SIG*; its features are described in the following subsections.

2.1 Extraction of WST Entities

The first step in using the information stored in service directories is to present a way to extract, represent, and manage the knowledge in it. To assist service developers, the system must determine which web services are related to currently under developing web service and how other service providers develop, enrich and document their web services.

For this reason, this paper presents a structure that stores this type of knowledge by interlinking web service entities based on semantic relations that may exist between them. Therefore, we must first formally identify the involved entities. Second, the semantic relations between these entities must be defined and formulated. Third, the instances of these relations in service directories must be discovered, and a method of representation must be presented.

Each service description involves a brief piece of documentation about its related web service and specifies the operations that the web service offers. Web services present one or more functions that are called operations. Operations may take some input from the user, create some process based on these input data, and produce some results as output. So, Four type of entities can be distinguished, namely, *Web Service*, *Operation*, *InputArgument*, and *OutputArgument* in web service descriptions. The ERD diagram of Figure 1 shows the relationships and cardinality between these entities.

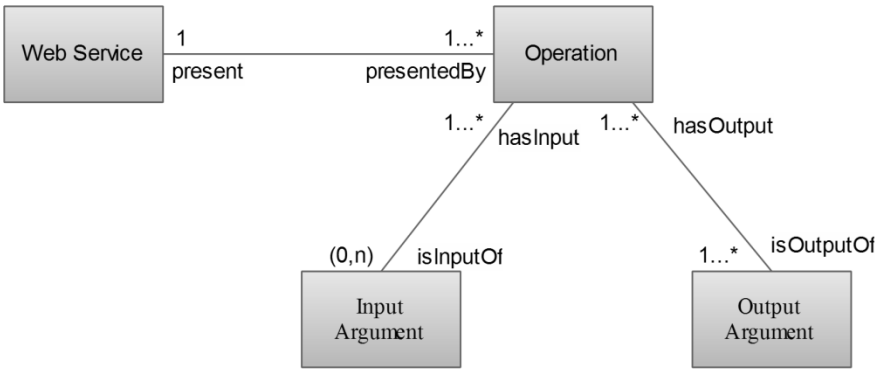

Figure 1. Entity-Relationship Diagram of WST Entities

A naming Schema is designed to name each entity based on its URL and the entity name. The following, show the designed identifiers for each of the entities described in Figure 1.

- *Service* URI: URL/Service Name
- *Operation* URI: URL/Service Name #Operation Name
- *OutputArgument* URI: URL/Service Name #Operation Name > Argument
- *InputArgument* URI: URL/Service Name #Operation Name < Argument

2.2 Constructing the SIG

In the previous section, the related entities were identified, and a unique identifier was defined for them to prevent ambiguity. In this section, an extension of the semantic relations introduced in (Fardin et al., 2012) are defined. By "semantic relation", we mean a relationship that is defined formally by using semantic technology.

Based on the ERD diagram in Figure 1, Table 1 present the semantic relations between four entities, namely, *Service*, *Operation*, *InputArgument* and *OutputArgument*. The set of *X* shows the possible semantic relations between these entities. This set is given by:

$$X = \{present,\ hasInput,\ hasOutput\} \bigcup \{presentedBy,\ isInputof,\ isOutputOf\} \quad (1)$$

and is shown as the union of two sets to clear the reversal of each semantic relation.

Table 1 shows a graphical representation, a short explanation and a real-world example for each of the defined semantic relations. All of the real-world examples were extracted from *Seekda*, a web service crawler available at http://webservices.seekda.com/.

Table 1. Relation between WST entities

Relation	Notation	Explanation	Real world Example
present	$s_i \rightarrow o_j$	the service s_i presents the operation o_j	WebserviceX.net /**globalweather**.asmx → GetCitiesByCountry
hasInput	$o_i \rightarrow a_j$	the operation o_i has an input argument named a_j	WebserviceX.net/**globalweather**.asmx#GetCitiesByCountry → CountryName
hasOutput	$o_i \rightarrow a_j$	the operation o_i has an output argument named a_j	WebserviceX.net/**globalweather**.asmx#GetCitiesByCountry → GetCitiesByCountryResponse

Some sort of semantic relations may exist between the instances of each entity. For example, the semantic relations that may exist between web services are introduced in (Fardin et al., 2012) by some real examples.

Based on the findings in (Fardin et al., 2012), by investigating real-world web services, the set of Δ shows the possible semantic relations between web services, and is given by:

$$\Delta = \{\text{is Similar To, is New Version Of, is Richer Than, is Composed Of}\} \cup \{\text{is Similar To, is Old Version Of, is Weaker Than, is In Plan Of}\} \quad (2)$$

Furthermore, each service presents one or more operations. Operations may have some sort of semantic relationship between them. Based on the findings in (Fardin et al., 2012), by investigating real-world web services, the set of ∇ shows the possible semantic relations between operations. We extend this set with two more relations shown in Table 2, and is given by:

$$\nabla = \{\text{is Similar To, has Same Result As, has Same Input As, is Richer Than, } \rightarrow \text{ can Be Linked To, can Assist, complement}\} \cup \{\text{is Similar To, has Same Result As, has Same Input As, is Weaker Than, can Get Input From, can Used By, complemented By}\} \quad (3)$$

Finally, each operation may have some input and at least one output argument. The extracted semantic relations between arguments are shown in Table 3.

The set of Λ showing the possible semantic relations between arguments of service operations is described in Table 3 and is given by:

$$\Lambda = \{\text{is Similar To, is Replacement Of, is Related To, is Richer Than}\} \cup \{\text{is Similar To, is Replacement Of, is Related To, is Richer Than}\} \quad (4)$$

Table 2. Extension of semantic relations between operations

Relation	**Explanation**	**Real-World Example**
canAssist	An operation can help another operation, e.g., by providing needed arguments or converting arguments to the desired format	webservicex.com/USZip.asmx#GetInfoByCity → flash-db.com/LocationByZipService.asmx#getDistanceByZip
complement	An operation provides a value-added feature to another operation.	WebserviceX.net/ConvertTemperature.asmx#ConvertTemp → ws.cdyne.com/WeatherWS/Weather.asmx# GetCityWeatherByZIP

Table 3. Relations between operation arguments

Relation	**Explanation**	**Real-world Example**
isSimilarTo	Two arguments indicate exactly the same concept	webservicex.com/USZip.asmx#globalweather # GetCitiesByCountry <CountryName → webservicex.com/USZip.asmx#globalweather #GetWeather<CountryName
isReplacementOf	Two arguments interchangeably indicate the same concept or thing and can be converted into one another	webservicex.com/USZip.asmx#GetInfoByCity< USCity → www.riptrails.com/ Weather # Get_Weather< zip
isRelatedTo	Two arguments are in the same domain	webservicex.com/globalweather# GetCitiesByCountry<CountryName → webservicex.com/USZip.asmx#GetInfoByCity< USCity

isRicherThan	The argument has more information than other arguments	webservicex.com/Weather#GetWeather> GetWeatherResult → deeptraining.com/Weather#GetWeather>GetWeatherResult

SIG is constructed using the defined semantic relations. Figure 2 shows an instance of this graph. For the sake of simplicity, all of the entities and relations are not shown in this figure. All entities in Figure 2 are real-world examples extracted from current service directories.

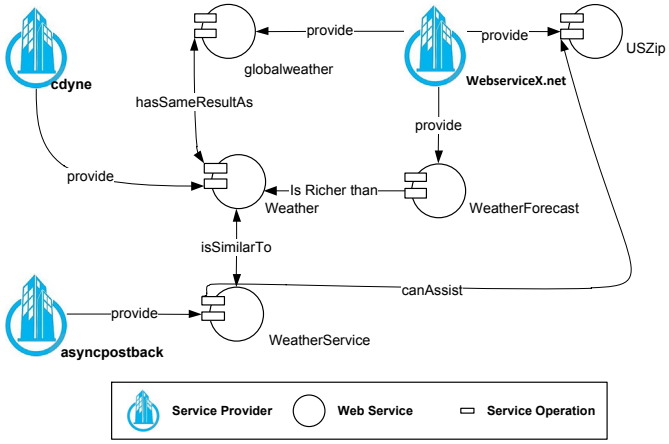

Figure 2. Graphic view of the semantic relations

2.3 Assistant System Architecture

Suppose some web services have been published in the network. The description of these services is preserved in the service directory. In order to create, preserve and use the *SIG*, certain processes have been created in the service directory. Figure 3 shows the general architecture of the system. The components of the assistant system are Ontology, Inference Engine, SPARQL Endpoint, RDF Repository, Monitoring Service, Matching Service, and service descriptions. The assistant unit is the access point for developers and helps them develop new web services.

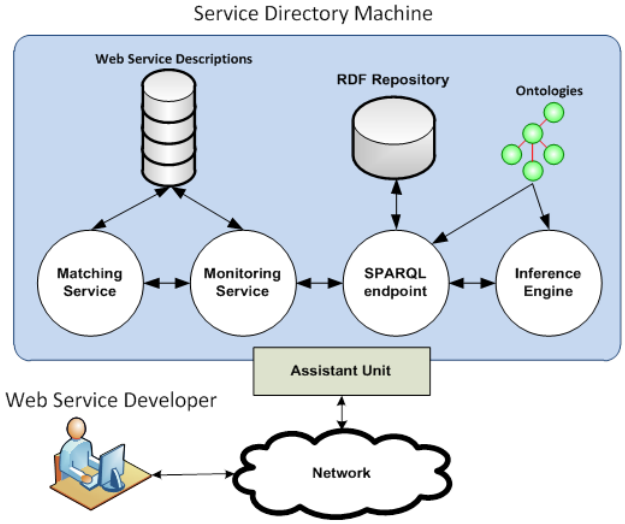

Figure 3. Architecture of the Assistant System

Each component of the system has been explained individually below:

Ontology: First, the semantic relations are defined formally by means of an ontology. With the aid of the

ontology, the entities or concepts, the semantic relations, the cardinality constraints and the mutual exclusiveness of the semantic relations are defined. The following are examples of a cardinality constraint: "An *Operation* is *presented by* only one *Service*" or "A *Service* may *present* one or more *Operation*". For an example of a mutually exclusive constraint, *isSimilarTo* and *isRicherThan* cannot be established between two operations at the same time. For this porpose, we extend OWLS-SD (Fardin et al., 2012) and develop it with new relations introduced in this paper. OWLS-SD is a semantic language based on RDF. The goal of OWLS-SD is to create machine-readable ontologies.

Matching Service: finding the similarity between arguments, operations and services in order to extract relevant information in the service directory, is the responsibility of this service. This process extracts the semantic relations between WST entities. This service is implemented as described in (Plebani & Pernici, 2009). The matching service returns the degree of similarity between two entities as a number in the interval [0,1].

RDF Repository: The relations founded by Matching Service are stored in RDF Repository. Similar to our previous work in (Fardin et al., 2012), this paper selects the RDF language as the language used to implement the *SIG*. RDF triples are simple but strong and efficient tools for encoding knowledge in semantic environments. Each RDF triple connects two entities by specifying the relationship between them. The relationships between entities are semantic relations that are defined by the constructed OWLS-SD ontology. By using RDF triples, the *SIG* is implemented in this RDF store.

Monitoring Process: Providers publish their services and register the service descriptions in the directory. To make these newly added services usable by the assistant unit; they must be added to the semantic interlinked graph. Discovering new registered services and joining these services to the graph is the responsibility of this process.

SPARQL Endpoint: In order to use constructed *SIG* in the presented tool and extract useful knowledge to assist developers, some functions should be defined and implemented in the proposed system. In these functions the RDF repository must be queried. A good choice to use for querying the RDF repository is the SPARQL language. SPARQL is a protocol and also a language to query RDF databases through the internet. In addition, SPARQL queries are human readable, and users can provide high-level queries in SPARQL to extract knowledge stored in the RDF repository. So in the ***SPARQL Endpoint***, we define and implement the required functions as follows.

First, a mathematical relation is defined that retrieves a set of services for a given service and a semantic relation. This relation can be defined as follows:

$$f : \Delta \times S \to \{S\} \tag{5}$$

where $S=\{s_1,s_2,s_3,...\}$ is the set of published web services in the network. By considering the semantic relation $\delta \in \Delta$ and a service $s \in S$, the relation f returns a set of services that have a semantic relation δ with s. For example, the statement $'Weatherservive' \in f(isSimilarTo,'Weather')$ declares that a service called $'Weather'$ has the *is Similar To* relation with a service called $'Weatherservive'$. A graphical representation of this statement is illustrated in Figure 4.

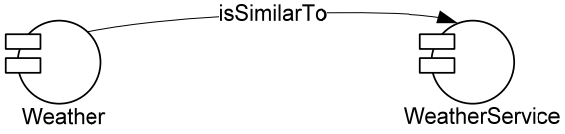

Figure 4. Graphical representation of $'Weatherservive' \in f(isSimilarTo,'Weather')$

Furthermore, such mathematical relations are defined to extract information about operations and arguments. For this purpose, the following mathematical relation is defined to extract information about a given operation and semantic relation:

$$g : \Lambda \times O \to \{O\} \tag{6}$$

where $O_i = \{o_{i1}, o_{i2}, ..., o_{in}\}$ is the finite set of operations presented by the service s_i and $O = \bigcup O_i$ is the set of all operations presented in the network. For example, by considering $\tau \in \Lambda$ and an operation $o \in O$, the relation g returns a set of operations that have a semantic relation τ with o.

As an example, the statemen $'GetWeatherData' \in g(isSimilarTo,'GetCityForecastByZip')$ declares that an operation called $'GetWeatherData'$ has the *isSimilarTo* relation with an operation called

'*GetCityForecastByZip*'. Figure 5 shows the graph representation of this instance.

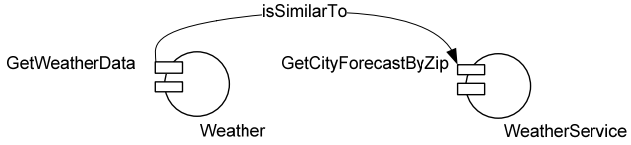

Figure 5. Graphical representation of $'GetWeatherData' \in g(isSimilarTo, 'GetCityForecastByZip')$

Finally, $I_{ij} = \{in_1, in_2, ..., in_l\}$ and $R_{ij} = \{out_1, out_2, ..., out_l\}$ are the finite set of input and output/result arguments, respectively, of operation o_j presented by service s_i, and $A = \{\bigcup I_{ij}\} \bigcup \{\bigcup R_{ij}\}$ is the set of the arguments of all the operations. The mathematical relation h is defined to extract information about a given argument and semantic relation:

$$h : \nabla \times A \rightarrow \{A\} \tag{7}$$

Inference Engine: This process is responsible for checking the constraints of the system based on the extended OWLS-SD ontology. Also, it tries to discover and establish new semantic relations based on the properties of them.

Assistant Unit: This component implements the procedures assisting developers by extracting and presenting useful information from the *SIG*. The assistant unit receives input from service developers by providing a web-based user interface.

The general activity of service directory is described in Algorithm 1 as follow:

Algorithm 1 Service directory process

1: Monitor the Web Service Descriptions database
2: **if** Monitoring Service find a new service description **then**
3: Use ***Matching Process*** to find possible semantic relations
4: Insert discovered semantic relations in the ***RDF Repository*** by the use of ***SPARQL Endpoint***
5: Use ***Inference Engine*** to check the Integrity of ***RDF Repository*** and discovery of possible semantic relations
6: **end if**

The start-up of the system and initialization of it is performed by this algorithm too.

2.4 WSDATool and Process of Assisting

The architecture of Assistant unit is depicted in figure 6. In general, Assistant unit mine the RDF repository to extract useful patterns and assist developers, based on these patterns and some guidance rules.

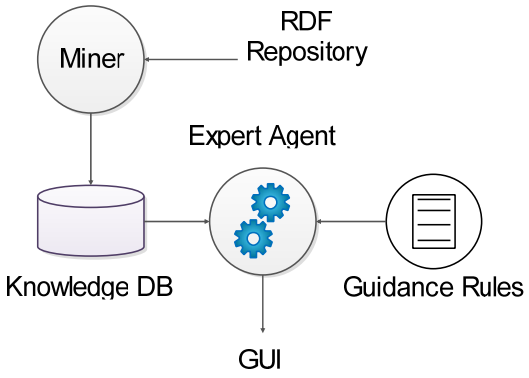

Figure 6. The WSDATool processes

In the following, the components of assistant unit are described:

Miner: this process mine the RDF triples stored in the RDF Repository to extract useful patterns of web service

designing. Rules that generated by this unit store in the Knowledge DB.

Knowledge DB: design patterns generated by Mine process are stored in this database. An example of such pattern is: most weather services get city name as input, most airport services get 3 digit airport codes as input.

Expert Agent: This process utilise Knowledge DB to assist web service developers using some guidance rules. These rules answer to the following questions:

- Which operations consume these defined inputs, and what is the output of these operations?
- Which operations generate these defined outputs, and what is the input of these operations?
- Which operations can be linked to this defined operation to make a composition plan?
- Which operations are similar to the developed operation?

For an example of using the semantic relations, the operations that have isRicherThan relations with a user-defined operation can be used to improve and enrich the written service. The canBeLinkedTo services can also be used to improve a being developed service to make it more contributable in composition plans.

GUI Agent: Service developers communicate to the GUI agent by means of a web-based GUI. This GUI consists of several tabs. In each tab, the developer obtains assistance from the assistant unit. In the proposed system, the process of assisting begins simultaneously with the process of developing a web service. There is no need to have an initial web service description file or documentation. Developers can start to write a new web service using the tool.

The assisting process for the developer takes place according to the following Algorithm and the sequence of operations is shown in Figure 7.

Algorithm 2 Assisting Steps

1: The developer defines input and output parameters.
2: The assistant tool recommends unambiguous parameter names if necessary.
3: The developer constructs input/output arguments.
4: The assistant tool verifies the messages to prevent redundancy and complexity.
5: The developer constructs the desired operations.
6: The assistant tool recommends decomposition or enrichment of operations.
7: The developer constructs the desired service.
8: The assistant tool recommends that the developer add or amend operations.
9: The assistant tool forces the developer to document the messages, operations, and service.
10: The developer can download the final service description and code template.

In each step, developers can see similar or richer entities extracted from *SIG* and are categorised by the tool in order to redefine them if necessary. Beside recommendation derived from *SIG*, the tool assist developers in some areas as follow:

Clear Naming: Defining the arguments of a web service in an easy-to-understand way is very important to service discovery. For example, in [1], it was shown that 35.9 percent of users querying the MSN search engine to find APIs used queries based on the name of the APIs, the methods and the data types. Therefore, the naming strategy and API design can improve the discovery of services. In (Juan Manuel et al., 2010), investigating some published services showed that approximately 82 percent of services suffer from name ambiguity.

Documentation: Furthermore, Good documentation can facilitate better discovery of provided services. In (Juan Manuel et al., 2010), the effect of good documentation in better service discovery has been shown. This work, by investigating some published services, shows that approximately 68 percent of them are not well documented.

Failure: Defining suitable error message to handle failures helps service consumers understand the source of the error in case of any failures and improves data quality.

Furthermore, users can use the annotating tools, such as Maleshkova et al. (2009) or Dietze et al. (2011), to construct a semantic annotating description for their being developed services. Making a composition plan is another feature of the tool. The user can construct a composition plan by specifying the input parameters and the output parameters. If possible, the tool tries to find a chain of one or more web services consuming input parameters and producing output parameters. For each step in which the tool cannot proceed, it asks the

developer to resolve the problem, if possible. The tool can also be used to assist service consumers in making queries. The output of the system can be used as a query to describe the consumer's needs. Table 4 shows aspects that the tool can assist developers.

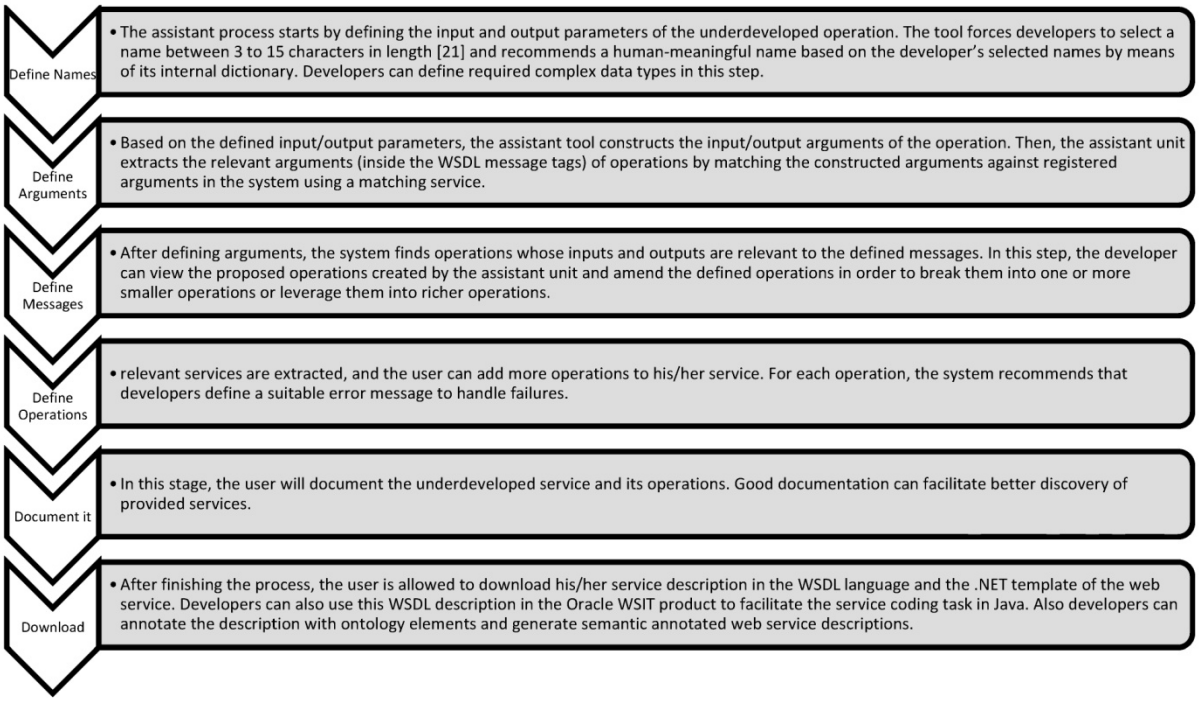

Figure 7. Overall process of the tool

Table 4. Categories that system recommends the developers

Assisting Aspect	**Explanation**
Naming	Unambiguous naming of inputs arguments, outputs arguments, and operators
Documentation	Adding documentation to service, arguments and port types
Adding new operations to the service	Adding value-added operations to the developing service
Enriching the service operations	Making service results more usable by producing more useful outputs
Detecting design defects	Handling redundant port-types and messages
Considering composition	Changing input or output arguments to high composition adaptability
Resolving complexity	Dividing a complex operation or argument into smaller one, make data types and messages simpler

3. Results and Discussion

In this section, some aspects of the implemented tool were examined through experimentation. To experiment with the tool, 558 web services were selected from Seekda. Services are chosen so that there exist some similarity between them in order to be able to construct a dense SIG graph. These services are divided into two sets named Directory_set, consisting of 538 services, and Experimental_set, consisting of the remainder of the services. Then, a semantic interlinked graph was constructed using Directory_set. To construct the framework, jUDDI is used as the service directory (downloadable at http://juddi.apache.org/) and Sesame as the SPARQL endpoint (downloadable at http://www.openrdf.org/), both with some modifications.

The monitoring process and the inference process were developed as a system process using the .Net framework. The monitoring process is an event-based process and is activated upon service insertion.

To experiment with the tool, the services in Experimental_set are redesigned using .Net web service developing IDE without considering the description of them. The service descriptions for the written services were gathered in a set called $\mathrm{Re}sult_{IDE}$. At the next step, the services are rewrite using the assistant tool. The service descriptions of the written services from this step were gathered in a second set called $\mathrm{Re}sult_{Tool}$. Therefore, we created three sets of service descriptions, as follows:

$$\mathrm{Re}sult_{IDE} = \{sri_1, sri_2, ...sri_{20}\}, \mathrm{Re}sult_{Tool} = \{srt_1, srt_2, ...srt_{20}\}, Experimental_{Set} = \{ses_1, ses_2, ...ses_{20}\}$$

We inspected different aspects of the usefulness of the tool: the size of the service description file, the amount of documentation included, the number of port types, and the number of failure-handling messages. More failure error handling messages, more documentation, a low number of port types, and a smaller size for the description file are desired for services. This is because these parameters influence service discovery precision and speed and also influence the readability of services. For each service, we calculated how the tool enhanced these parameters. Let us define the following measures:

- *CalculateSize*(s): Get a service description file and return the size in bytes.
- *DocumentationNumbers(s)*: Return the amount of documentation in the service description normalised by the number of defined entities inside the service according to the ERD diagram of Figure 1.
- *PortNumbers(s)*: Return the number of defined port types of the service *s*.
- *FailureMessages(s)*: Return the number of defined failure-handling messages of the service s.

The enhancement made by using the tool for a service ses_i could be normalised by the following formula:

Description size enhancement:

$$\frac{CalculateSize(ses_i) - CalculateSize(srt_i)}{CalculateSize(ses_i)} \times 100 \tag{8}$$

Documentation enhancement:

$$\frac{DocumentationNumbers(srt_i) - DocumentationNumbers(ses_i)}{DocumentationNumbers(ses_i)} \times 100 \tag{9}$$

Ports enhancement:

$$\frac{PortNumbers(ses_i) - PortNumbers(srt_i)}{PortNumbers(ses_i)} \times 100 \tag{10}$$

Failure-handling enhancement:

$$\frac{FailureMessages(ses_i) - FailureMessages(srt_i)}{FailureMessages(ses_i)} \times 100 \tag{11}$$

The number of port types, amount of documentation, and failure-handling messages are calculated in this way, and the results of using the tool are depicted in Figure 8. Each figure represents the enhancement made by the tool with respect to a given aspect. In each figure, the enhancements for all 20 web services are shown. The horizontal axes are web services and the vertical axes are the normalised values of the performance parameter.

As shown in Figure 8, in all aspects investigated, the tool was shown to be useful. In failure message enhancements, a zero value means that appropriate failure messages are embedded by the provider of the service. A zero value in port-type enhancements means the provider used the minimum required port-types, so developers could not minimise it.

We constructed several queries using the tool to evaluate the effect of the system on service discovery. For this purpose, the refined services in the $\mathrm{Re}sult_{Tool}$ set were returned to the original data set that was collected from *Seekda*. The experiment shows that the degree of similarity between created queries and services in $\mathrm{Re}sult_{Tool}$ is greater than the degree of similarity between created queries and services in $Experimental_{Set}$.

Finally, in the worst case in which no similar arguments, operations or services exist for our being developed services, the system can be useful in developing a more unambiguous, well-documented, compact and

anti-pattern–free web service.

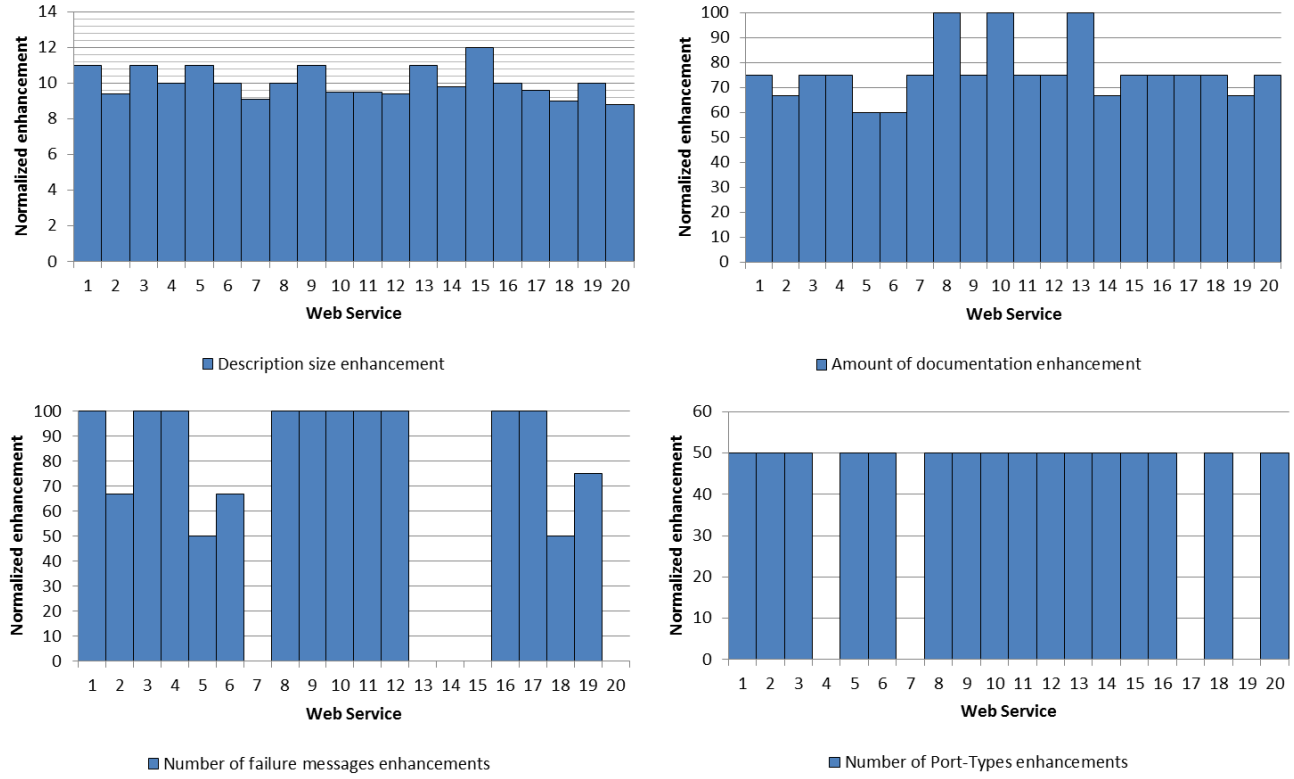

Figure 8. Normalised enhancements of the tool for the description file size, amount of documentations, number of failure messages, and number of port types

4. Future Works

In this paper, a graph of WST entities is constructed. These entities are interlinked based on the semantic relations that are defined between them. This graph of entities is implemented by RDF triples that make it possible to interlink all WST entities around the world. The heart of the system is the ontology, which makes it possible for all processes to have the same understanding of the network. RDF triples use this ontology to express the relation between entities.

Using the presented knowledge and the provided method of knowledge management, an assistant system for web service developers was designed and implemented. By using this system, developers can use patterns that have previously been developed by providers from around the world to develop powerful and easy-to-discover services. Our experiments showed that the presented tool can help service and software developers enrich their work. By using this system, the developed services will have a high-quality description, and this will make it easier to discover them.

Although different description languages are presented so far such as SAWSDL (Iqbal et al., 2008), OWLS(Klusch & Kapahnke, 2009; Matthias, Benedikt, & Katia 2009), and WSMO (Maleshkova et al., 2009), and some methods that try to presents unified methods of presenting web services (Cardoso, Barros, May, & Kylau, 2010; Loutas, Peristeras, & Tarabanis, 2011; Pilioura & Tsalgatidou, 2009), we present our method by using WSDL 1.1 description language. But, our method can be applying to other description languages as well by slights modifications to implemented codes. WSDL 1.1 is selected because current web services developing IDE use this description language.

Automatic or semi-automatic integration with service repositories are main necessary feature that must be considered in future works.

References

Bhaskar, M., Claudia, N., Avare, S., Claudio, M., & J., N. E. (2004). An Architecture for Recommendation Based

Service Mediation. *Semantics of a Networked World, LNCS, 3226*, 250-262.

Birukou, A., Blanzieri, E., D'Andrea ,V., Giorgini, P., & Kokash, N. (2007). Improving Web Service Discovery with Usage Data. *IEEE Software, 24*(6), 47-54. http://dx.doi.org/10.1109/MS.2007.169

Bizer, C., Heath, T., & Berners-Lee, T. (2009). Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems (IJSWIS), 5*(3), 1-22. http://dx.doi.org/10.4018/jswis.2009081901

Blake, M. B., & Nowlan, M. F. (2007). *A Web Service Recommender System Using Enhanced Syntactical Matching.* Paper presented at the IEEE International Conference on Web Services (ICWS 2007), Salt Lake City, Utah, USA. http://dx.doi.org/10.1109/ICWS.2007.28

Cardoso, J., Barros, A., May, N., & Kylau, U. (2010). *Towards a Unified Service Description Language for the Internet of Services: Requirements and First Developments*.

Crasso, M., Mateos, C., Zunino, A., & Campo, M. (2010). EasySOC: Making Web Service Outsourcing Easier. *Information Sciences, in press.* http://dx.doi.org/10.1016/j.ins.2010.01.013

Crasso, M., Rodriguez, J. M., Zunino, A., & Campo, M. (2010). Revising WSDL documents: Why and How. *IEEE Internet Computing, in press*. http://dx.doi.org/10.1109/MIC.2010.81

Crasso, M., Zunino, A., & Campo, M. (2008). Easy web service discovery: A query-by-example approach. *Sci. Comput. Program., 71*(2), 144-164. http://dx.doi.org/10.1016/j.scico.2008.02.002

Dietze, S., Yu, H. Q., Pedrinaci, C., Liu, D., & Domingue, J. (2011). *SmartLink: a Web-based editor and search environment for Linked Services*. Paper presented at the 8th Extended Semantic Web Conference (ESWC 2011), Heraklion, Greece.

Dong, X., Halevy, A., Madhavan, J., Nemes, E., & Zhang, J. (2004). *Similarity search for web services.* Paper presented at the Thirtieth international conference on Very large data bases, Toronto, Canada.

Fardin, A. M., Naser, N. B., & Ali, N. M. (2012). Empower service directories with knowledge. *Know.-Based Syst., 30*, 172-184. http://dx.doi.org/10.1016/j.knosys.2012.01.010

Hoffmann, R., Fogarty, J., & Weld, D. S. (2007). *Assieme: finding and leveraging implicit references in a web search interface for programmers.* Paper presented at the 20th annual ACM symposium on User interface software and technology.

Ichii, M., Hayase, Y., Yokomori, R., Yamamoto, T., & Inoue, K. (2009). *Software component recommendation using collaborative filtering.* Paper presented at the ICSE Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation, Vancouver, BC. http://dx.doi.org/10.1109/SUITE.2009.5070014

Iqbal, K., Sbodio, M. L., Peristeras, V., & Giuliani, G. (2008, December 03-December 05). *Semantic Service Discovery using SAWSDL and SPARQL.* Paper presented at the Fourth International Conference on Semantics, Knowledge and Grid. http://dx.doi.org/10.1109/SKG.2008.87

Juan Manuel, R., Marco, C., Alejandro, Z., & Marcelo, C. (2010). Improving Web Service descriptions for effective service discovery. *Science of Computer Programming, 75*(11), 1001-1021. http://dx.doi.org/10.1016/j.scico.2010.01.002

Juan Manuel Rodriguez, M. C., Zunino, A., & Campo, M. (2010). Improving Web Service Descriptions for effective service discovery. *Science of Computer Programming, in press*.

Klusch, M., & Kapahnke, P. (2009). *OWLS-MX3: An Adaptive Hybrid Semantic Service Matchmaker for OWL-S.* Paper presented at the International Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web (SMR2-2009), Washington D.C., United States.

Loutas, N., Peristeras, V., & Tarabanis, K. (2011). Towards a reference service model for the Web of Services. *Data & Knowledge Engineering, 70*(9), 753-774. http://dx.doi.org/10.1016/j.datak.2011.05.001

Lu, X., Lin, J., Zou, Y., Peng, J., Liu, X., & Zha, L. (2010). *Investigating, Modeling, and Ranking Interface Complexity of Web Services on the World Wide Web*. Paper presented at the Proceedings of the 26 010th World Congress on Services. http://dx.doi.org/10.1109/SERVICES.2010.58

Maleshkova, M., Pedrinaci, C., & Domingue, J. (2009). *Supporting the creation of semantic RESTful service descriptions*. Paper presented at the 8th International Semantic Web Conference (ISWC 2009), Washington D.C., USA.

Mateos, C., Crasso, M., Zunino, A., & Campo, M. (2010). An Evaluation on Developer's Acceptance of EasySOC:A Development Model for Service-Oriented Computing *Proceedings of the 11th Argentine*

Symposium on Software Engineering Simposio Argentino de Ingenieria de Software (ASSE2010) - 39th JAIIO.

Matthias, K., Benedikt, F., & Katia, S. (2009). OWLS-MX: A hybrid SemanticWeb service matchmaker for OWL-S services. *Web Semantics: Science, Services and Agents on theWorldWideWeb, 7*, 121-133.

Pilioura, T., & Tsalgatidou, A. (2009). Unified publication and discovery of semantic Web services. *ACM Trans. Web, 3*(3), 1-44. http://dx.doi.org/10.1145/1541822.1541826

Plebani, P., & Pernici, B. (2009). URBE: Web Service Retrieval Based on Similarity Evaluation. *IEEE Transaction on Knowledge and Data Enginering, 21*(11), 1629-1643. http://dx.doi.org/10.1109/TKDE.2009.35

Robillard, M. P., Walker, R. J., & Zimmermann, T. (2010). Recommendation Systems for Software Engineering. *Software, IEEE, 27*(4), 80 - 86. http://dx.doi.org/10.1109/MS.2009.161

Rodriguez, J. M., Crasso, M., Zunino, A., & Campo, M. (2009). Discoverability anti-patterns: frequent ways of making undiscoverable Web Service descriptions *Proceedings of the 10th Argentine Symposium on Software Engineering (ASSE2009) - 38th JAIIO* (pp. 1-15). Mar del Plata, Argentina.

Rodriguez, J. M., Crasso, M., Zunino, A., & Campo, M. (2010). Automatically detecting opportunities for Web Service descriptions improvement *Proccedings of the 10th IFIP Conference on e-Business, e-Services, and e-Society (I3C 2010*).

Sellami, M., Tata, S., Maamar, Z., & Defude, B. (2009). *A Recommender System forWeb Services Discovery in a Distributed Registry Environment.* Paper presented at the Fourth International Conference on Internet and Web Applications and Services, Venice/Mestre, Italy.

Shripad, M. U., & V., P. T. (2005). *Dynamic Selection ofWeb Services with Recommendation System.* Paper presented at the International Conference on Next Generation Web Services Practices, Seoul, Korea.

Sneed, H. M. (2010). *Measuring web service interfaces*. Paper presented at the WSE.

Vu, L.-H., Hauswirth, M., Porto, F., & Aberer, K. (2006). A search engine for QoS-enabled discovery of semantic web services. *International Journal of Business Process Integration and Management, 1*(4), 244-255. http://dx.doi.org/10.1504/IJBPIM.2006.012623

Zheng, Z., Ma, H., Lyu, M. R., & King, a. I. (2009). *WSRec: A Collaborative Filtering Based Web Service Recommender System.* Paper presented at the IEEE International Conference on Web Services, Los Angeles, CA. http://dx.doi.org/10.1109/ICWS.2009.30