ISSN 2320 -2602

International Journal of Advances in Computer Science and Technology (IJACST), Vol.2 , No.8, Pages : 18-23 (2013)
Special Issue of ICCECT 2013 - Held during 16-17 August, 2013, Thailand

Enhancing Structural and Semantic Similarity Evaluation for Web Service Retrieval

Nitipan Pompan[1], Twittie Senivongse[2]
[1]Department of Computer Engineering, Chulalongkorn University, Bangkok, Thailand, nitipan.p@student.chula.ac.th
[2]Department of Computer Engineering, Chulalongkorn University, Bangkok, Thailand, twittie.s@chula.ac.th

Abstract : Web services have been used widely in modern software applications since they, as networked software units, provide certain functionality that can be incorporated into building software applications in a flexible manner. Like other software, Web services may experience changes and failures which make them inaccessible to service consuming applications. In this case, it is then necessary for those applications to find other alternative services. One of the effective approaches is to evaluate both structural similarity and semantic similarity between the description of the service in use and those of other candidate services in order to identify an alternative. This paper follows an approach called URBE to determine structural and semantic similarity between Web services. In particular, we enhance the evaluation on data type similarity, by also considering family of data types and covariance/contravariance principle, and on name similarity, by also considering text similarity. The enhanced algorithm is called M-URBE. An experiment shows that, in comparison with URBE, M-URBE can improve the performance of Web service retrieval.

Key words : Ontology, Retrieval, Web services, WSDL

INTRODUCTION

Web services have been used widely in modern software applications since they, as networked software units, provide certain functionality that can be incorporated into building software applications in a flexible manner. Web Services technology is built upon a number of standards. Two of them that are relevant to this paper are the Web Service Description Language (WSDL) and the Universal Description Discovery and Integration (UDDI). WSDL [1] is a language used by a service provider for describing the Web service interface (or portType), service operations, input and output messages of the operations which contain data elements of different types, and how to access the service. To become known to service consumers, the service provider can make the service-related information, including the WSDL, available through a service discovery mechanism such as a service registry or search engine. UDDI [2] is a form of a registry service that allows service providers to publish business and Web service information, and allows service consumers to look up the providers and their WSDL information before selecting and engaging a Web service.

Like other software, Web services may experience changes and failures which make them inaccessible to service consuming applications. In this case, it is then necessary for service consumers to find other alternative services for their applications. Researchers have proposed different approaches to discover Web services that are similar to the one requested by a consumer, i.e. search by keywords, search by structural similarity, and search by semantic similarity.

The search-by-keyword approach enhances a general mechanism of a search engine. For example, Hatzi et al. [3] provide a specialized search engine that specifically crawls the Web for WSDL documents and semantic specifications of Web services and builds an enhanced indexing and retrieval mechanism. Elgazzar et al. [4] can boost keyword search for Web services by mining WSDL documents to cluster them into functionally similar service groups first.

The second approach discovers Web services by structural similarity. Plebani and Pernici [5] argue that search capability of UDDI is limited such that it provides search by name or category of providers' business and services but does not exploit the content of WSDL documents during retrieval. They propose an algorithm called UDDI Retrieval by Example (URBE) which uses UDDI as its service registry and analyzes the structure of WSDL documents and the names or terms defined inside them in order to find the services with the structure similar to the one queried by a consumer. The evaluation compares sets of service operations by comparing input and output messages of the operations, which, in turn, is based on comparison of types of the data contained by the messages. Similarly, Stroulia and Wang [6] use structural matching in their work to compare operation signatures in two WSDL documents.

The third approach considers semantic similarity. Plebani and Pernici [5] and Stroulia and Wang [6] also use WordNet [7] to determine linguistic similarity of names in the WSDL documents. In addition, using the SAWSDL mechanism [8], Plebani and Pernici consider similarity of ontological terms that are annotated to different parts of the WSDL structure in their URBE algorithm. Liu et al [9] argue that the names specified in a WSDL document are not isolated in meaning but have semantic connections that associate them together to describe the service function. Hence, they employ search results from Web search engine as a context for calculating semantic distance of any two names from the two compared services. Service similarity is measured upon these distances.

This paper follows the URBE approach since it is comprehensive in terms of utilizing both syntactic and semantic contents of service descriptions. Nevertheless, we identify three drawbacks regarding similarity of data types and of names in WSDL documents:

(1) In data type comparison, different data types in the

ISSN 2320 -2602

International Journal of Advances in Computer Science and Technology (IJACST), Vol.2 , No.8, Pages : 18-23 (2013)
Special Issue of ICCECT 2013 - Held during 16-17 August, 2013, Thailand

same family are not differentiated. For example, URBE considers all types in the Integer family – long, int, short, and byte – as the same type and a perfect match with each other.

(2) In data type comparison, the principle of contravariant input and covariant output is not employed for data type compatibility [10]. That is, URBE does not consider that the input type of a provider's service can be more generalized than, and still be compatible with, the input type of the query. It does not consider either the reverse where the output type of a service can be more specialized than, and still be compatible with, the output type of the query.

(3) In name comparison, meaning of names (or terms) is considered. However, names defined for operations and data elements in WSDL documents may not be full dictionary words and hence not be included in WordNet. URBE does not consider string similarity of the textual names.

It is seen that these drawbacks are likely to lower down the efficiency of Web service retrieval. We present the M-URBE as a modification to the URBE algorithm on the aforementioned aspects, and report on its performance.

The next section of this paper gives an overview of service similarity comparison in M-URBE, followed by a section giving the detail of the algorithm. Then, an experiment on the performance of M-URBE in comparison with URBE is presented. The final section concludes the paper with future work.

OVERVIEW OF SIMILARITY COMPARISON FOR WEB SERVICES

In M-URBE, similarity between a service queried by a consumer (*Sq*) and a provided Web service (*Sp*) is determined by comparing the set of operations and related data elements (or parameters) of the query and that of the provided service. The comparison is pairwise, i.e. portType-to-portType, operation-to-operation, input-to-input, and output-to-output comparison. Fig. 1 depicts the similarity matching process of URBE which is adopted by M-URBE. The process makes use of the following functions:

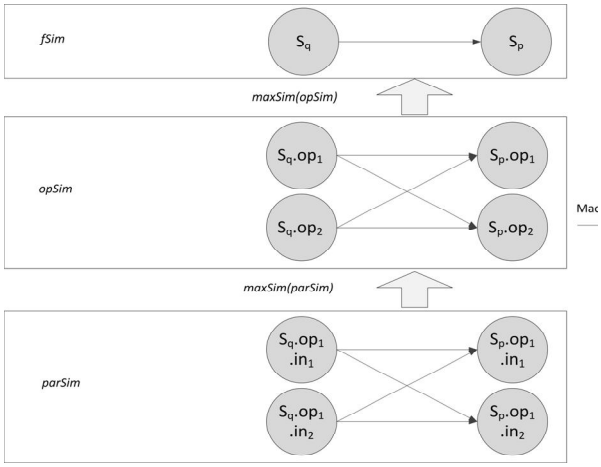


Fig 1: Similarity matching process [5]

- *parSim* is a bottom-level function which gives a similarity score for each pair of input parameters and each pair of output parameters. The score is based on similarity of parameter types which can be either *simpleType* or *complexType*, and is calculated by a function *datatypeSim*. In the case of *complexType*, similarity of type names is additionally considered and computed by a function *nameSim*, which, in turn, considers similarity of terms that constitute each name by using a function *termSim*. Since we compute a similarity score for every possible pair of input (or output) parameters, we consider the pairs that yield a maximum similarity score. A function *maxSim* is used to determine the maximum score that represents the similarity score of input (or output) parameters for a pair of operations being compared.
- *opSim* is a mid-level function which gives a similarity score for each pair of operations. The score is based on similarity of input parameters and of output parameters using *parSim*, and similarity of operation names using *nameSim* (and hence *termSim*). Again, since we compute a similarity score for every possible pair of operations, we use *maxSim* to determine the maximum score that represents the similarity score of operations for a pair of portTypes being compared.
- *fSim* is a top-level function which gives a similarity score for a queried service and a provided service. The score is based on similarity of operations using *opSim* and similarity of portType names using *nameSim* (and hence *termSim*).

M-URBE

This section describes the detail of URBE along with the enhancement that M-URBE introduces to the algorithm. An example of a query for a *PolicyServiceSoap* as in Fig. 2 is used in the explanation.

```
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
…
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:tns="http://tempuri.org/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:sawsdl="http://www.w3.org/ns/sawsdl">
    <wsdl:types>
        <s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
            <s:element name="SearchPolicy">
                <s:complexType>
                    <s:sequence>
                        <s:element minOccurs="0" name="request" type="tns:PolicyRequest" />
                    </s:sequence>
                </s:complexType>
            </s:element>
            <s:complexType name="PolicyRequest"
sawsdl:modelReference="http://127.0.0.1/ontology/insurance.owl# Request ">
                <s:sequence>
                    <s:element minOccurs="1" name="PolicyNumber" type="s:string" />
                    <s:element minOccurs="1" name="ReferenceNumber" type="s:int" />
                </s:sequence>
            </s:complexType>
            ……..
        </s:schema>
    </wsdl:types>
    <wsdl:message name="SearchPolicySoapIn" >
        <wsdl:part name="parameters" element="tns:SearchPolicy" />
    </wsdl:message>
    <wsdl:message name="SearchPolicySoapOut" >
        <wsdl:part name="parameters" element="tns:SearchPolicyResponse" />
    </wsdl:message>
    <wsdl:portType name="PolicyServiceSoap"
sawsdl:modelReference="http://127.0.0.1/ontology/insurance.owl#PolicyInquiryService">
        <wsdl:operation name="SearchPolicy"
sawsdl:modelReference="http://127.0.0.1/ontology/insurance.owl#PolicySearching">
            <wsdl:input message="tns:SearchPolicySoapIn" />
            <wsdl:output message="tns:SearchPolicySoapOut" />
        </wsdl:operation>
    </wsdl:portType>
    ...
</wsdl:definitions>
```

Fig 2: Example of a queried service *PolicyServiceSoap*

ISSN 2320 -2602

International Journal of Advances in Computer Science and Technology (IJACST), Vol.2 , No.8, Pages : 18-23 (2013)
Special Issue of ICCECT 2013 - Held during 16-17 August, 2013, Thailand

As with URBE, in similarity evaluation, a queried service and a provided service will be represented by an abstract notation, independent of WSDL versions, as follows:

- σ_i = *(name,{op})* represents a portType of a Web service with a name and a set of operations.
- *op* = *(name,{input$_m$,output$_n$},modelReference)* represents an operation with a name and a set of *m* input and *n* output parameters, together with a *modelReference* that refers to a semantic term of an ontology which is annotated to this operation using the SAWSDL mechanism.
- *input* = *(name,type,modelReference)* represents an input parameter with a name, data type, and annotated ontological term denoting semantics of the input.
- *output* = *(name,type,modelReference)* represents an output parameter with a name, data type, and annotated ontological term denoting semantics of the output.

The query in Fig. 2 can be represented by an abstract notation as in Fig. 3. Given a provided service in Fig. 4, we will demonstrate the evaluation of their structural and semantic similarity.

Structural Similarity Evaluation

A. Maximization Function (maxSim)

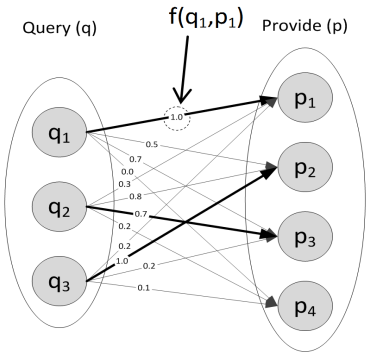
As mentioned earlier, for a particular WSDL element (i.e. parameters, operations, and terms within names), the computation of a similarity score compares every possible pair of WSDL elements of the query (i.e. q_i) and the counterpart WSDL elements of the provided service (i.e. p_i) as in Fig. 5. We use the maximum score to represent the similarity score. The *maxSim* function (1) is taken from [11]:

```
σ.name = PolicyServiceSoap
σ.op1 = {
    σ.op1.name = SearchPolicy,
    σ.op1.inputPar1 = {
        σ.op1.inputPar1.name = request,
        σ.op1.inputPar1.type = PolicyRequest
        σ.op1.inputPar1.modelReference =
http://127.0.0.1/ontology/insurance.owl#Request
        σ.op1.inputPar1.input1 = {
            σ.op1.inputPar1.input1.name = PolicyNumber,
            σ.op1.inputPar1.input1.type = string
        }
        σ.op1.inputPar1.input2 = {
            σ.op1.inputPar1.input2.name = ReferenceNumber,
            σ.op1.inputPar1.input2.type = int
        }       …
}
```

Fig 3: Abstract description of a queried service *PolicyServiceSoap*

```
σ.name = ContractInquiryService
σ.op1 = {
    σ.op1.name = InquiryContract,
    σ.op1.inputPar1 = {
        σ.op1.inpuPar1.name = contractRequest,
        σ.op1.inputPar1.type = ContractRequest
        σ.op1.inputPar1.modelReference =
http://127.0.0.1/ontology/insurance.owl#PolicyRequest
        σ.op1.inputPar1.input1 = {
            σ.op1.inputPar1.input1.name = PolicyNumber,
            σ.op1.inputPar1.input1.type = long
        }
        σ.op1.inputPar1.input2 = {
            σ.op1.inputPar1.input2.name = ReferenceNumber,
            σ.op1.inputPar1.input2.type = short
        }…
}
```

Fig 4: Abstract description of a provided service *ContractInquiryService*



$maxSim(f(q,p)) = (f(q_1,p_1)+f(q_2,p_3)+f(q_3,p_2))/4 = (1.0+0.7+1.0)/4 = 0.675$

Fig 5: All possible matching between elements in the sets *q* and *p*, modified from [5]

$$maxSim(f(q,p)) = \frac{max(\sum_{j=1..|p|}^{i=1..|q|} f(q_i, p_j))}{|p|} \quad (1)$$

where q = set of elements in query and $q_i \in q$,
p = set of elements in provided service and $p_i \in p$,
f = any similarity function (i.e. *parSim*, *opSim*, *termSim*); f is in [0..1], and
maxSim is in [0..1].

For example, assume that Fig. 5 represents matching between a set of operations (*q*) in a query and a set of operations (*p*) in a provided service. $opSim(q_i,p_j)$ would be used in place of $f(q_i,p_j)$ in (1) to determine similarity between operations q_i and p_j. The similarity score of these two sets of operations is 0.675.

B. Name Similarity Function (nameSim)

To determine similarity between any textual names, we use the *nameSim* function (2) [5]:

$$nameSim(n_q, n_p) = maxSim(termSim(\{t_{q,i}\},\{t_{p,j}\})) \quad (2)$$

where n_q = a name in query, comprising a set of terms $t_{q,i}$,
n_p = a name in provided service, comprising a set of terms $t_{p,j}$, and
termSim = similarity function for each pair of terms.

Any textual name will be tokenized into terms by using the rules [5] in Table 1. Then we determine similarity for each pair of terms. URBE uses the linguistic similarity score from WordNet [7] for the function *termSim*. We enhance the function by also considering string similarity since tokenized terms may not be full dictionary words and not be included in WordNet, e.g. the term *Num* is not in WordNet but it should get some similarity score when being compared to the term *Number*. String similarity is represented by Levenshtein distance between two terms [12], i.e. the minimum number of single-character edits (insertion, deletion, substitution) required to change one term into the other. Our function *termSim* is defined in (3):

Table 1: Tokenization rules for names [5]

Rule	Name	Tokenized term
Case change	PolicyNumber	policy, number
Underscore elimination	Policy_Number	policy, number
Suffix number elimination	PolicyNumber1	policy, number

ISSN 2320 -2602

International Journal of Advances in Computer Science and Technology (IJACST), Vol.2 , No.8, Pages : 18-23 (2013)
Special Issue of ICCECT 2013 - Held during 16-17 August, 2013, Thailand

$$termSim(\{t_{q,i}\},\{t_{p,j}\}) = \quad (3)$$
$$Weight_{WordNet} \cdot termSim_{WordNet}(\{t_{q,i}\},\{t_{p,j}\})$$
$$+Weight_{Levenshtein} \cdot termSim_{Levenshtein}(\{t_{q,i}\},\{t_{p,j}\})$$

where $termSim_{WordNet}$ = similarity score by WordNet in [0,1]

$termSim_{Levenshtein}$ = similarity score by Levenshtein distance, which is normalized to [0,1] where 1 means identical term, and

$Weight_{WordNet} + Weight_{Levenshtein} = 1$.

For example, to compare similarity between the input parameter name *Request* in Fig. 3 and *ContractRequest* in Fig. 4, we obtain $\{t_{q,i}\} = \{request\}$ and $\{t_{p,j}\} = \{contract, request\}$. We then calculate *termSim* for every possible pair of terms, using WordNet and Levenshtein distance. Suppose $Weight_{WordNet}$ = 0.7 and $Weight_{Levenshtein}$ = 0.3. Matching between $t_{q,1}$ = *request* and $t_{p,2}$ = *request* will give the maximum WordNet score and maximum Levenshtein score (both are 1), whereas $t_{p,1}$ = *contract* is left unmatched. Using *nameSim* (2) which uses *maxSim* (1), *nameSim*(*Request*, *ContractRequest*) yields (1+0)/2 = 0.5.

C. Data Type Similarity Function (datatypeSim)

XML schema data types [13] are used to define types of operation parameters and can be either *simpleType* or *complexType* (which itself can be further expanded to a number of *simpleTypes*). As mentioned earlier, URBE does not differentiate data types in the same family (Fig. 6), e.g. int matches long in the same manner as int matches int. Hence we enhance by applying the concept of Generalizable Nominal Attribute (GNA) [14] which determines similarity between concepts within a hierarchy by their distance.

In addition, we consider the principle of contravariant input and covariant output [10]. The input type of the provided service which is more generalized than the input type of the query is considered compatible and will also get higher similarity score than the case of the provided service's input type that is more specialized. The reverse applies for the output type. That is, the output type of the provided service can be more specialized than the output type of the query and will get higher similarity score than the case of the provided service's output type that is more generalized.

Table 2 shows similarity scores for *simpleTypes*. Fundamentally the scores are inversely proportional to the information loss that will occur if we apply a casting from *q* to *p* [5], but for Integer and Real types, GNA scores apply. The GNA scores for types in the same family are in Table 3 and Table 4.

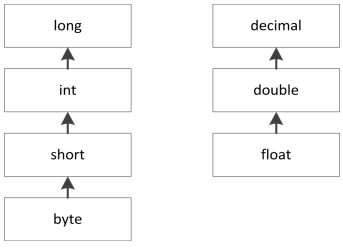


Fig 6: Integer and Real families

Table 2: Similarity scores for simpleTypes, modified from [5]

Provide (p) \ Query (q)	Integer	Real	String	Date	Boolean
Integer	$GNA_{Integer}(q,p)$	0.5	0.3	0.1	0.1
Real	1.0	$GNA_{Real}(q,p)$	0.1	0.0	0.1
String	0.7	0.7	1.0	0.8	0.3
Date	0.1	0.0	0.1	1.0	0.0
Boolean	0.1	0.0	0.1	0.0	1.0

Table 3: $GNA_{Integer}$ scores for Integer family

Provide (p) \ Query (q)	Input				Output			
	long	**int**	**short**	**byte**	**long**	**int**	**short**	**byte**
long	1	1	1	1	1	0.5	0.33	0.25
int	0.5	1	1	1	1	1	0.5	0.33
short	0.33	0.5	1	1	1	1	1	0.5
byte	0.25	0.33	0.5	1	1	1	1	1

Table 4: GNA_{Real} scores for Real family

Provide (p) \ Query (q)	Input			Output		
	decimal	**double**	**float**	**decimal**	**double**	**float**
decimal	1	1	1	1	0.5	0.3
double	0.5	1	1	1	1	0.5
float	0.33	0.5	1	1	1	1

Also, in the case that a parameter of the provided service is optional (minOccurs=0) and is not matched to any parameter of the query, its data type similarity score would be 1.

The *datatypeSim* function is shown in Fig. 7. The function *simpletypeScore* refers to the score obtained from Tables 2-4. For example, in Fig. 3, suppose ele_q1.name = *PolicyNumber*, ele_q1.dt = string, ele_q2.name = *ReferenceNumber*, ele_q2.dt = int. In Fig. 4, suppose ele_p1.name = *PolicyNumber*, ele_p1.dt = long, ele_p2.name = *ReferenceNumber*, ele_p2.dt = short. Therefore, *datatypeSim*(*PolicyNumber*, *PolicyNumber*) is 1*0.3 = 0.3, and *datatypeSim*(*ReferenceNumber*, *ReferenceNumber*) is 1*0.5 = 0.5.

D. Parameter Similarity Function (parSim)

To determine similarity of parameters, both parameter names and parameter types are considered. The similarity functions for input parameters and output parameters are defined in (4) and (5) respectively [5]:

$$parSim_{input}(op_q.input, op_p.input)$$
$$= Weight_{namePar} \cdot nameSim(op_q.input.name, op_p.input.name)$$
$$+ Weight_{typePar} \cdot datatypeSim(op_q.input.type, op_p.input.type) \quad (4)$$

$$parSim_{output}(op_q.output, op_p.output)$$
$$= Weight_{namePar} \cdot nameSim(op_q.output.name, op_p.output.name)$$
$$+ Weight_{typePar} \cdot datatypeSim(op_q.output.type, op_p.output.type) \quad (5)$$

```
function datatypeSim(ele_q, ele_p)
  if(ele_q.dt is simpleType and ele_p.dt is simpleType)
    return nameSim(ele_q.name, ele_p.name)*
           simpletypeScore(ele_q.dt, ele_p.dt)
  else if (ele_q.dt is complexType and
           ele_p.dt is complexType)
      return nameSim(ele_q.name, ele_p.name)*
    datatypeSim (ele_q.dt.elements, ele_p.dt.elements)
  else if ele_p.dt is Optional and
          ele_p is not matched by any ele_q
        return 1
      else
        return 0
end function
```

Fig 7: Pseudocode of *datatypeSim*, modified from [5]

where $Weight_{namePar} + Weight_{typePar} = 1$ and

parSim is in [0,1].

E. Operation Similarity Function (opSim)

To determine similarity of operations, both operation names and input/output parameters are considered. The similarity function for operations is defined in (6) [5]:

$$\begin{aligned} &opSim(op_q, op_p) = \\ &Weight_{OperationName} \cdot nameSim(op_q.name, op_p.name) \\ &+ Weight_{Par} \cdot [0.5 \cdot maxSim(parSim_{input}(op_q.\{input\}, op_p.\{input\})) \\ &+ 0.5 \cdot maxSim(parSim_{output}(op_q.\{output\}, op_p.\{output\}))] \end{aligned} \quad (6)$$

where $Weight_{OperationName} + Weight_{Par} = 1$ and
opSim is in [0,1].

F. portType Similarity Function (fSim)

To determine similarity of the queried service and the provided service, their portType names and operations are considered. The similarity function for portTypes is defined in (7) [5]:

$$\begin{aligned} &fSim(\sigma_q, \sigma_p) = Weight_{portTypeName} \cdot nameSim(\sigma_q.name, \sigma_p.name) \\ &+ Weight_{Operations} \cdot maxSim(opSim(\sigma_q.\{op\}, \sigma_p.\{op\})) \end{aligned} \quad (7)$$

where $Weight_{portTypeName} + Weight_{Operations} = 1$ and
fSim is in [0,1].

Semantic Similarity Evaluation

URBE can use semantic similarity evaluation instead of structural similarity evaluation. The idea is the same in that it still considers the structure of a WSDL document, but the evaluation is on the semantic terms that are annotated to different parts of the WSDL structure by using SAWSDL [8]. The semantic terms are ontological terms in a service domain ontology to which the WSDL document refers by using *modelReference*. When the queried service and provided service are semantically annotated by terms from the same domain ontology, we can determine their similarity.

For semantic similarity evaluation, the function *annSim* defined in (8) [5] is used in place of *nameSim* in functions (4)-(7) above:

$$annSim(a_q, a_p) = \begin{cases} pathSim(a_q, a_p), & a_q, a_p \text{ are both classes/properties} \\ classPropSim(a_q, a_p), & a_q \text{ is class, } a_p \text{ is property} \\ propClassSim(a_q, a_p), & a_q \text{ is property, } a_p \text{ is class} \end{cases} \quad (8)$$

where a_q = annotation in the queried service,
a_p = annotation in the provided service, and
annSim is in [0,1].

A. Class-Class and Property-Property Similarity (pathSim)

The function *pathSim* of URBE takes into account the subsumption path which connects the two classes or two properties in the service domain ontology as defined in (9) [5]. In addition, we employ the principle of contravariant input and covariant output when annotations are associated with inputs and outputs, and the additional functions are defined in (10) and (11):

$$pathSim(a_q, a_p) = \begin{cases} 1 & Exact \\ 0 & Failed \\ \dfrac{1}{pathlength(a_q, a_p) + 1} & a_q \text{ related to } a_p \end{cases} \quad (9)$$

$$pathSim_{parInput}(a_q, a_p) = \begin{cases} 1 & Exact \text{ or } a_q \subseteq a_p \\ 0.4 & Partial \\ 0 & Failed \\ \dfrac{1}{pathlength(a_q, a_p) + 1} & a_p \subseteq a_q \end{cases} \quad (10)$$

$$pathSim_{parOutput}(a_q, a_p) = \begin{cases} 1 & Exact \text{ or } a_p \subseteq a_q \\ 0.4 & Partial \\ 0 & Failed \\ \dfrac{1}{pathlength(a_q, a_p) + 1} & a_q \subseteq a_p \end{cases} \quad (11)$$

where $pathlength(a_q, a_p)$ = the number of hops constituting
the longest path connecting a_q and a_p,
$a_p \subseteq a_q = a_p$ is subsumed by (more specialized than)
a_q, and
$a_q \subseteq a_p = a_p$ subsumes (more generalized than) a_q.

For example, in Fig. 3, a_q refers to *Request* and is associated with the input parameter *request* of the query. In Fig. 4, a_p refers to *PolicyRequest* and is associated with the input parameter *contractRequest* of the provided service. If *PolicyRequest* is a direct subclass of *Request* in the ontology (i.e. a_p is more specialized than a_q), *pathSim*(*Request*, *PolicyRequest*) = 1/(1+1) = 0.5, by using (10).

B. Class-Property Similarity (classPropSim)

When a_p is one of the properties of class a_q, similarity is proportional to the number of properties that class a_q has. The function *classPropSim* is defined in (12) [5]:

$$classPropSim(a_q, a_p) = \begin{cases} \dfrac{1}{\text{number of properties of } a_q}, & a_p \text{ is property of class } a_q \\ 0, & a_p \text{ is not property of class } a_q \end{cases} \quad (12)$$

C. Property-Class Similarity (propClassSim)

When a_q is one of the properties of class a_p, similarity is 1 since a_p has the queried property (and more). The function *propClassSim* is defined in (13) [5]:

$$propClassSim(a_q, a_p) = \begin{cases} 1, & a_q \text{ is property of class } a_p \\ 0, & a_q \text{ is not property of class } a_p \end{cases} \quad (13)$$

EXPERIMENTAL RESULTS

We evaluate the performance of M-URBE in comparison with that of URBE. The benchmark used for the evaluation is SAWSDL-TC [15] which comprises 1,080 WSDL documents semantically annotated by SAWSDL. Each WSDL document belongs to one of nine domains: communication, economy, education, food, geography, medical, simulation, travel, and weapon. The benchmark provides 42 WSDL documents that are used as queries, each of which also belongs to one of the nine domains. We randomly select five of them to use as queried services in the experiment, i.e.

- *1personbicyclecar_price_service.wsdl*
- *bookpersoncreditcardaccount__service.wsdl*
- *citycountry_hotel_service.wsdl*
- *surfinghiking_destination_service.wsdl*

- *userscience-fiction-novel_price_service.wsdl*

These queried services are used for retrieving provided services with similar structure and semantics, i.e. those that give the *fSim* value not less than a similarity threshold (between 0 and 1). That is, if the *fSim* value of a provided service is less than the threshold, it will not be returned as a result. The performance measurement is by F-Measure in (14), and we use the average of the F-Measures obtained from all queries to represent the performance of both algorithms.

$$F - Measure = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \qquad (14)$$

where *Precision* = no. of returned services in the same domain as the query / no. of returned services
Recall = no. of returned services in the same domain as the query / total no. of services in the same domain as the query.

Considering structural similarity, we run two experiments for both algorithms. The first one evaluates the effect of the modification on *datatypeSim*, and the second one on *nameSim*. In the first experiment, we set $Weight_{Levenshtein}$ to 0, and identify the best values of $Weight_{portTypeName}$, $Weight_{OperationName}$, and $Weight_{NamePar}$ by varying these parameters to achieve all weight combinations using the values in the set {0.1, 0.3, 0.5, 0.7, 0.9}. The best result giving the best F-Measure values is when $Weight_{portTypeName}$ = 0.9, $Weight_{OperationName}$ = 0.7, and $Weight_{NamePar}$ = 0.9, as shown in Fig. 8(a). In the second experiment, we keep these best weights and vary $Weight_{Levenshtein}$ by using the values in {0, 0.1, 0.3, 0.5, 0.7, 0.9}. We have found that $Weight_{Levenshtein}$ does not help improve F-Measure values.

Considering semantic similarity, we run a similar experiment but $Weight_{Levenshtein}$ is not relevant since *annSim* is used in place of *nameSim*. The best result giving the best F-Measure values is when $Weight_{portTypeName}$ = 0.1, $Weight_{OperationName}$ = 0.1, and $Weight_{NamePar}$ = 0.1, as shown in Fig. 8(b).

It can be seen that in both graphs, M-URBE can improve the performance of service retrieval when the threshold is around 0.5-0.8 which we consider to be practical for use since very low similarity threshold can give too many returned services whereas very high similarity threshold may give too few services for selection.

CONCLUSION

We present M-URBE as an enhancement to URBE with regard to similarity evaluation of data types and of textual names. The experimental results show that different compatibility level of types in the same family and contravariance/covariance compatibility can help improve the performance of service retrieval even though string name similarity measure does not show strong impact on the performance. This may be the case of the services in the benchmark being named properly using complete words and hence WordNet can already serve the purpose. We plan to build a service retrieval tool upon the M-URBE algorithm. The tool should also be able to recommend modification or mapping that needs to be applied to the retrieved services to enable seamless substitution for a queried service.

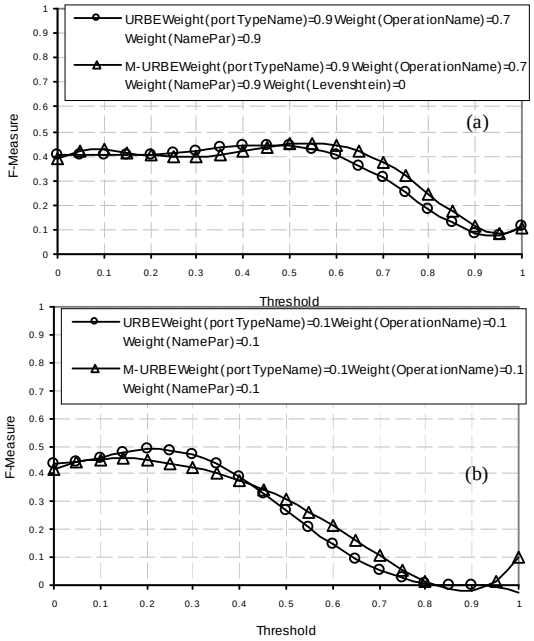


Fig 8: Performance of similarity evaluation (a) structural (b) semantic

REFERENCES

[1] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. (March 2001). Web services description language (WSDL) 1.1. [Online]. Available: http://www.w3.org/TR/wsdl

[2] OASIS. Universal description discovery and integration. [Online]. Available: http://uddi.xml.org/

[3] O. Hatzi, G. Batistatos, M. Nikolaidou, and D. Anagnostopoulos, "A specialized search engine for Web service discovery," in *Proc. 2012 IEEE 19th Int. Conf. Web Services (ICWS 2012)*, Hawaii, USA, 2012, pp. 448-455.

[4] K. Elgazzar, A. E. Hassan, and P. Martin, "Clustering WSDL documents to bootstrap the discovery of Web services," in *Proc. 2010 IEEE Int. Conf. Web Services (ICWS 2010)*, Florida, USA, 2010, pp. 147-154.

[5] P. Plebani and B. Pernici, "URBE: Web service retrieval based on similarity evaluation," *IEEE Trans. Knowledge and Data Engineering*, vol. 21, no. 11, pp. 1629-1642, November 2009.

[6] E. Stroulia and Y. Wang, "Structural and semantic matching for assessing web-service similarity," *Int. J. Cooperative Information Systems*, vol. 14, no. 4, pp. 407–437, 2005.

[7] Princeton University. WordNet a lexical database for English. [Online]. Available: http://wordnet.princeton.edu/

[8] J. Farrell and H. Lausen. (August 2007). Semantic annotations for WSDL and XML schema. [Online]. Available: http://www.w3.org/TR/sawsdl/

[9] F. Liu, Y. Shi, J. Yu, T. Wang, and J. Wu, "Measuring similarity of Web services based on WSDL," in *Proc. 2010 IEEE Int. Conf. Web Services (ICWS 2010)*, Florida, USA, 2010, pp. 155-162.

[10] G. Castagna, "Covariance and contravariance: conflict without a cause," *ACM Trans. Programming Languages and Systems (TOPLAS)*, vol. 3, no. 17, pp. 431-447, May 1995.

[11] V. Andrikopoulos and P. Plebani, "Retrieving compatible Web services," in *Proc. 2011 IEEE Int. Conf. Web Services (ICWS 2011)*, Washington, USA, pp. 179-186.

[12] G. Navarro, "A guided tour to approximate string matching," *ACM Computing Surveys (CSUR)*, vol. 33, no. 1, pp. 31 – 88, March 2001.

[13] P.V. Biron and A. Malhotra. (October 2004). XML schema part 2: datatypes second edition. [Online]. Available: http://www.w3.org/TR/xmlschema-2/

[14] X. Shen, X. Jin, R. Bie, and Y. Sun, "MSC: a semantic ranking for hitting results of matchmaking of services," in *Proc. 30th Annu. Int. Computer Software and Applications Conf. (COMPSAC 2006)*, Chicago, USA, 2006, pp. 291-296.

[15] SemWebCentral. SAWSDL-TC. [Online]. Available: http://projects.semwebcentral.org/projects/sawsdl-tc/