

Algorithms for Web Services Discovery and Composition Based on Syntactic and Semantic Service Descriptions

Seog-Chan Oh, Hyunyoung Kil, Dongwon Lee, and Soundar R. T. Kumara

The Pennsylvania State University, PA 16802, USA
{seogchan, hkil, dongwon, skumara}@psu.edu

Abstract

As the number of available web services is rapidly increasing, the effective and efficient service discovery and composition becomes a pressing problem for value-added distributed applications. To address this problem, in this paper, we present efficient service discovery and composition algorithms that exploit both syntactic and semantic service descriptions of web services.

1. Introduction

In this paper, we present our solution submitted to the IEEE CEC/EEE 2006 WS-Challenge [1] where participants are to develop efficient service discovery and composition solutions. This year, solutions need to address both *syntactic* and *semantic* matching based on the service descriptions and XML types in WSDL.

The syntactic competition assumes that the matching of services is based on the string equivalence of parameter names of the input and output messages of a service. On the other hand, the semantic competition adopts the idea of so called *Semantic Web Services* that represent web services with semantic descriptions about the interface and its characteristics. For the semantic description purpose, the competition offers a type hierarchy in XML Schema, and provides type definitions about the input and output parameters of the web services considered. Consequently, the semantic competition extends the matching of parameter names to the matching of parameter types defined by XML Schema type hierarchy.

Although our solutions consider both syntactic and semantic matching together, for the simpler presentation, two matching are discussed in separate sections.

2. Syntactic Matching

Suppose that a web service, w , has typically two sets of parameters: $w^i = \{I_1, I_2, \dots\}$ for SOAP request (as input) and $w^o = \{O_1, O_2, \dots\}$ for SOAP response (as output). When w is invoked with all input parameters provided, w^i , w returns the output parameters, w^o . We assume that in order to invoke w , all input parameters in w^i must be provided (i.e., w^i are mandatory).

Consider a request r that has initial input parameters r^i and desired output parameters r^o . The **Web Service Discovery (WSD)** problem is to find a set of web services $w \in W$ such that (1) $r^i \supseteq w^i$ and (2) $r^o \subseteq w^o$. With a simple look-up table, in general, the WSD problem can be easily solved. Therefore, in the remainder of this section, we focus on the case in which no single web service fully satisfies the request r and thus one has to “compose” multiple web services, referred to as the **Web Service Composition (WSC)** problem. Whether the problem is WSD or WSC, note that the matching is solely based on the syntactic similarity of parameter names.

Let us denote the entire set of web services as W , and their parameters as P . Then, we can define the syntactic WSC problem as the AI planning problem with a state space $\Psi = \langle S, s_0, S_G, \Omega(\cdot), c \rangle$ [3] where:

- (1) The states $s \in S$ are collection of parameters in P
- (2) The initial state $s_0 \in S$ such that $s_0 = r^i$
- (3) The goal state $S_G \in S$ such that $r^o \subseteq S_G$
- (4) $\Omega(s)$ is the set of web services $w \in W$ such that $w^i \subseteq s$. That is, w can be invoked or applicable in the state s
- (5) $c(w)$ is the invocation cost of w

Then, based on $\Psi = \langle S, s_0, S_G, \Omega(\cdot), c \rangle$, we can formally define the WSC problem as follows:

Definition 1 (Syntactic Web Service Composition)

Given a request r with initial input parameters r^i

and desired output parameters r^o , The syntactic Web Service Composition (WSC) problem is to find a finite sequence of web services, w_1, w_2, \dots, w_n such that (1) w_i can be invoked sequentially from 1 to n , (2)

$(r^i \cup w_1^o \dots \cup w_n^o) \supseteq r^o$, and (3) the total cost $\sum_{i=1}^n c(w_i)$

is minimized.

Input: r^i, r^o **Output:** $PD_{ws}^{syn}(p)$

- 1: $s = (r^i \setminus r^o); C = \emptyset; d=1$
- 2: while $\neg(s \supseteq r^o)$ do
- 3: $\delta = \{w \mid w \in \Omega(s), w \notin C\}$
- 4: for each p in $w^o (w \in \delta)$
- 5: if $g_{r^i}(p) = \infty$
- 6: $g_{r^i}(p) = d; PD_{ws}^{syn}(p) = w; s = s \cup \{p\}$
- 7: $C = C \cup \delta; d++$

Figure 1. Forward search for syntactic composition

According to the competition description [1], all requests in syntactic competition can be satisfied by chaining web services in such a way that a predecessor web service can fully match the successor web service. This fact enables a highly specialized planning algorithm such as forward searching algorithm characterized by $g_{r^i}(p)$ – the cost of achieving $p \in P$ starting from a state r^i .

$$g_{r^i}(p) = \min_{w \in Ow(p)} [c(w) + \max_{p' \in w^o} g_{r^i}(p')] \quad (1)$$

Where, $c(w)$ is the invocation cost of a web service and it is assumed to be 1. $Ow(p)$ is a set of web services $w \in W$ such that $p \in w^o$. At first, $g_{r^i}(p)$ are initialized to 0 if $p \in r^i$ and to ∞ otherwise. Then, the current information state s is set as r^i . For $\forall w \in \Omega(s)$, each parameter $p \in w^o$ is added to s and $g_{r^i}(p)$ is updated until for $\forall p \in r^o$, $g_{r^i}(p)$ are obtained. If $\Omega(s) = \emptyset$, no solution exists. We refer to a web service, w , as a predecessor web service of $p \in P$ if w is the first web service to generate p . $PD_{ws}^{syn}(p)$ is an inverted index – for each parameter p , one can retrieve all web services that contain p . Once we obtain $PD_{ws}^{syn}(p)$ by the procedure of Figure 1, we can trace a

sequence of web services from r^o to r^i backward with guidance of $PD_{ws}^{syn}(p)$, resulting in a set of web services with minimum cost that satisfy the given request.

3. Semantic Matching

The semantic service discovery and composition focus on the matching of parameter types defined by the XML Schema type hierarchy¹. Let us denote a set of types as T . Then, a binary function $\tau: P \rightarrow T$ returns a corresponding type of the parameter P . If a type t_1 is inherited from a type t_2 , then t_1 is a subtype of t_2 and denoted by $t_1 \Vdash t_2$. Conversely, t_2 is a supertype of t_1 and denoted by $t_1 \Vdash t_2$.

Suppose that a web service w has one parameter in each of input and output parameter set, that is $w^i = \{I_1\}$ and $w^o = \{O_1\}$. Let w_1^{i1} and w_2^{o1} denote the I_1 and O_1 respectively. For $w_1, w_2 \in W$, if $\tau(w_1^{i1}) = \tau(w_2^{o1})$, it is evident that w_1 can invoke w_2 . Besides, we assume that if $\tau(w_1^{o1}) \Vdash \tau(w_2^{i1})$, then w_1 can invoke w_2 . That is, if the output parameter's type of w_1 is a subtype of the input parameter's type of w_2 , then w_1 can invoke w_2 .

```

<complexType name="Address">
  <sequence>
    <element name="name" type="string"/>
    <element name="street" type="string"/>
    <element name="city" type="string"/>
  </sequence>
</complexType>

<complexType name="US-Address">
  <complexContent>
    <extension base="Address">
      <sequence>
        <element name="state" type="US-State"/>
        <element name="zip" type="positiveInteger"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

```

Figure 2. "Address" and its subtype "US-Address"

Example: Let us consider two type examples as shown in Figure 2. For $w_1, w_2 \in W$, we can assume that $w_1^o = \{customerAddress\}$ and $w_2^i = \{clientAddress\}$

¹ Note that the "semantic matching" in WS-Challenge 2006 is only limited to the type compatibility, and does not consider "ontology matching" as in RDF and OWL. Our solution is customized toward for WS-Challenge specification, but is to be extended.

where $\tau(w_1^{o1}) = \text{US-Address}$ and $\tau(w_2^{i1}) = \text{Address}$. In this scenario, w_1 can invoke w_2 because $\text{US-Address} \Vdash \text{Address}$ (i.e., US-Address is a subtype of Address).

Suppose that a request r has the initial input parameter type $\tau(r^i)$ and desired output parameter type $\tau(r^o)$. Then, the Semantic Web Service Discovery (WSD) problem is to find a set of web services $w \in W$ such that (1) $\tau(r^{i1}) = \tau(w^{i1})$ or $\tau(r^{i1}) \Vdash \tau(w^{i1})$ and (2) $\tau(r^{o1}) = \tau(w^{o1})$ or $\tau(r^{o1}) \Vdash \tau(w^{o1})$.

We can solve the semantic WSD problem using a look-up table which can be constructed such that the supertype or subtype checking is computed at the cost of time complexity of $O(1)$. Similar to the syntactic WSC problem, we can define the semantic WSC problem as the AI planning problem in a state space $\Theta = \langle S, s_0, S_G, \Omega(\cdot), c \rangle$ where:

- (1) The states $s \in S$ are collection of types in T
- (2) The initial state $s_0 \in S$ is such that $s_0 = \tau(r^i)$
- (3) The goal state $S_G \in S$ is: $\exists t \in S_G$ such that $t = \tau(r^o)$ or $t \Vdash \tau(r^o)$
- (4) $\Omega(s)$ is the set of web services $w \in W$ such that $\exists t \in s$ $t = \tau(w^{i1})$ or $t \Vdash \tau(w^{i1})$. That is, w can be invoked or applicable in the state s
- (5) $c(w)$ is the invocation cost of w

Note that the main difference between Ψ of syntactic WSC and Θ of semantic WSC is their state spaces. That is, Ψ has parameter space as its search space while Θ has type definition space as its search space.

Definition 2 (Semantic Web Service Composition)

Suppose that a request r has initial input parameter type $\tau(r^i)$ and desired output parameter type $\tau(r^o)$.

Semantic Web Service Composition (WSC) is to find a finite sequence of web services, w_1, w_2, \dots, w_n such that

- (1) w_i can be invoked sequentially from 1 to n , (2)

$\tau(w_n^o) = \tau(r^o)$ or $\tau(w_n^o) \Vdash \tau(r^o)$, and (3) the total cost $\sum_{i=1}^n c(w_i)$ is minimized.

The structure of the algorithm is the same as the syntactic WSC. We define a forward searching

algorithm characterized by $g_{\tau(r^i)}(t)$ – the cost of achieving $t \in T$ starting from a state $\tau(r^i)$.

$$g_{\tau(r^i)}(t) = \min_{w \in Ow(t)} [1 + g_{\tau(r^i)}(\tau(w^{i1}))] \quad (2)$$

Where, $Ow(t)$ is a set of web services, $w \in W$ such that $\tau(w^{o1}) = t$ or $\tau(w^{o1}) \Vdash t$. At first, $g_{\tau(r^i)}(t)$ are initialized to 0 if $\tau(r^i) = t$ or $\tau(r^i) \Vdash t$. Otherwise they are initialized to ∞ . Then, the current information state s is set as $\tau(r^i)$. For $w \in \Omega(s)$, each $\tau(w^{o1})$ is added to s and $g_{\tau(r^i)}(t)$ is updated until the goal is obtained. If $\Omega(s) = \emptyset$, no solution exists. We name a web service w as a predecessor web service of $t \in T$ if w is the first web service to generate t . The algorithm is outlined in Figure 3.

Input: r^i, r^o **Output:** $PD_{ws}^{sem}(t)$

- 1: $s = \tau(r^i); C = \emptyset; d = 1$
- 2: while $\neg (\exists t \in s \text{ s.t. } t = \tau(r^o) \text{ or } t \Vdash \tau(r^o))$ do
- 3: $\delta = \{w \mid w \in \Omega(s), w \notin C\}$
- 4: for each w in δ
- 5: $t = \tau(w^{o1})$
- 6: if $g_{\tau(r^i)}(t) = \infty$
- 7: $g_{\tau(r^i)}(t) = d; PD_{ws}^{sem}(t) = w; s = s \cup \{t\}$
- 8: $C = C \cup \delta; d++$

Figure 3. Forward search for semantic composition

4. Conclusion

Our solution for WS-Challenge 2006 is presented in this paper. Ours is based on AI planning framework, and is able to address web service discovery and composition based on both syntactic and semantic service descriptions. In future, we will extend the type matching algorithm to adopt more complex methods (e.g., tree isomorphism, tree mapping, or approximate tree edit distance) and ontology matching as in RDF and OWL.

5. References

- [1] IEEE Joint Conference on E-commerce Technology (CEC'06) and Enterprise Computing, E-Commerce and E-Services (EEE'06). <http://insel.flp.cs.tu-berlin.de/wsc06/>
- [2] C. T. Kwok and D. S. Weld. "Planning to gather information". In *Proc. of AAAI*. Portland, OR, USA, 1996.
- [3] S. J. Russell and P. Norvig. "Artificial Intelligence: A Modern Approach", 2nd edn, Prentice-Hall, NJ, USA, 2002.