Semantic Web-Service Discovery and Composition Using Flexible Parameter Matching

Seog-Chan Oh, Jung-Woon Yoo, Hyunyoung Kil, Dongwon Lee, and Soundar R. T. Kumara
The Pennsylvania State University, University Park, PA 16802, USA
{seogchan, jwyoo, hkil, dongwon, skumara}@psu.edu

Abstract

When there are a large number of web services available and no single service satisfies the given request, one has to compose multiple web services to fulfill the goal, considering syntactic and semantic aspects. To address the web service composition issue, in this paper, we present a new composition algorithm by extending our previous work, WSRP (Web-service Planner) so as to determine relations between different parameter types during the process of service composition.

1. Introduction

This paper is intended for the 4th CEC/EEE07 WS-Challenge [1] where participants are required to address syntactic and semantic composition of web service chain based on the Web Services Description Language (WSDL) as well as the usage of OWL ontologies and XML Schema. The syntactic composition assumes that matching of services is based on the string equivalence of parameter names of the input and output messages of a service. Meanwhile, the semantic composition incorporates the use of OWL ontologies and XML Schema to define services and their relationships to each other. For example, the semantic competition will offer a type hierarchy defined in XML Schema or a class inheritance graph in OWL in order to provide the type definitions about the input and output parameters of the web services considered. Consequently, the semantic composition extends the matching of parameter names to the matching of parameter types defined in the type hierarchy information during the process of service composition.

This paper extends our previous work WSPR [3] which narrowly focused on the syntactic web-service composition so as to conform to requirements and specifications of this semantic-focused competition.

2. Flexible matching framework

Suppose that a web service w, has two sets of parameters: $w^i = \{I_1, I_2, ...\}$ for SOAP request (as input) and $w^o = \{O_1, O_2, ...\}$ for SOAP response (as output). When w is invoked with all input parameters, w^i, it 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). Suppose that P denotes a set of parameters and W denotes a set of web services.

When the "meanings" of two parameters, $p_1 \in P$ and $p_2 \in P$, are interchangeable, in general, they are said to be "matching" each other. The simplest way to check this is if two parameters have the *same* name and type: ($p_1.name = p_2.name$) $\wedge$ ($p_1.type = p_2.type$). Since web services are designed and created in isolation, however, this naive matching is often too rigid and thus misses cases like p_1 = ("password", string) and p_2 = ("passwd", string). To deal with this issue, we propose a generic Boolean function, match (p_1, p_2), that determines if two parameters p_1 and p_2 are matching or not. Formally,

Definition 1 (type-match) A boolean function, type-match ($p_1.type$, $p_2.type$), returns True if: (1) $p_1.type = p_2.type$, or (2) $p_1.type$ is derived from $p_2.type$ in a type hierarchy.

Definition 2 (match) A boolean function, match (p_1, p_2), returns True if: $p_1.name = p_2.name$, and type-match($p_1.type$, $p_2.type$) = True or if: $p_1.name \neq p_2.name$, but type-match($p_1.type$, $p_2.type$) = True

Definition 3 (Parameter Matching) When a boolean function, match(p_1, p_2), returns True, it is said that a parameter p_1 **matches** a parameter p_2.

The 9th IEEE International Conference on E-Commerce Technology and The 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services(CEC-EEE 2007)
0-7695-2913-5/07 $25.00

Throughout this paper, we will consider that if p_1 matches p_2 then $p_1 = p_2$. Therefore, in the process of finding applicable web services in certain information status, the value of output parameter p_2 is considered to satisfy the required value of input parameter p_1 as long as p_1 matches p_2.

Example: Let us consider two type examples as shown in Figure 1. For $w_1, w_2 \in W$, we can assume that w_1^o ={*customerAddress*} and w_2^i ={*clientAddress*} where *customerAddress.type* = "US-Address" and *clientAddress.type*="Address". In this scenario, w_1 can invoke w_2 according to our flexible parameter-matching framework because (1) *customerAddress.name* ≠ *clientAddress.name* but (2) type-match (*customerAddress, clientAddress*) = True ■

```
<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 1. "Address" and its subtype "US-Address"

With this flexible parameter matching framework, we can define the web-service composition problem formally.

Definition 4 (Semantic Web-Service Composition) Suppose that a request r has initial input parameters r^i and desired output parameters r^o. Semantic Web Service Composition (WSC) is defined as to find a finite sequence of web services, $w_1, w_2, ..., w_n$ such that (1) w_i can be invoked sequentially from 1 to n, using the flexible parameter-matching framework (2) $(r^i \bigcup w_1^o \ldots \bigcup w_n^o) \supseteq r^o$, and (3) the total cost $\sum_{i=1}^{n} c(w_i)$ is minimized.

3. Semantic Web-service composition

We can define the semantic WSC problem as a AI planning problem in a state space $\Psi = < S, s_0, S_G, \Omega(.), c >$ [2] where:

(1) The states $s \in S$ are collection of parameters in P

(2) The initial state $s_0 \in S$ is such that $s_0 = r^i$

(3) The goal state $S_G \in S$ is 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 using the flexible parameter-matching framework.

(5) $c(w)$ is the invocation cost of w

The basic concept of the semantic WSC algorithm comes from our previous work WSPR [4], but additionally we extend it from using the simple parameter name matching to the flexible parameter matching framework as presented in the section 2.

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

1: $s = (r^i \setminus r^o)$; $C = \phi$; 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: $g_{r^i}(p) = d$; $PD_{ws}^{syn}(p) = w$; $s = s \bigcup \{p\}$

7: $C = C \bigcup \delta$; d++

Figure 2. Forward search for composition

According to the competition description [1], all requests 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.

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

where, $c(w)$ is an 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 $\Omega(s) = \phi$, no solution exists. We name a web service w as a predecessor web service of $p \in P$ if w is the first web service to generate p. We denote $PD_{ws}^{syn}(p)$ to be an inverted index that contains the set of predecessor web services of p (i.e., token is p and documents are web services in the context of database community). Once we obtain $PD_{ws}^{syn}(p)$ by the procedure described in Figure 1, we can trace a sequence of web services from r^o to r^i backward with guidance of $PD_{ws}^{syn}(p)$.

4. Implementation

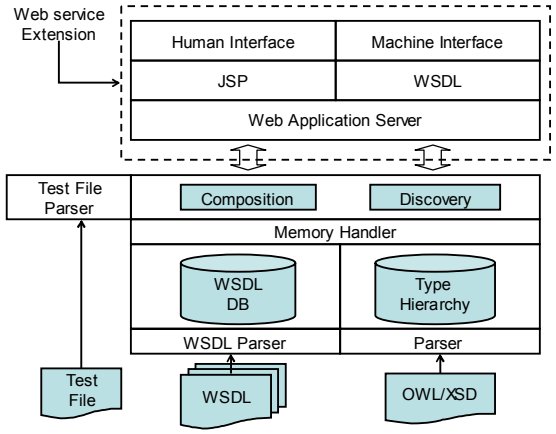

Figure 3. Overview of Implementation

Figure 3 illustrates the overview of our implementation for the 4th CEC/EEE07 WS-Challenge. Our implementation is based on our previous work, WSRP as shown in Figure 4. However, we extend WSPR based on the lessons we had learned from the last-year competition, where we experienced that the algorithm is required to efficiently handle I/O, memory, and computational resources. Therefore, we decide to build an adaptive memory DB and handler which can parses all the XML files including WSDL, XSD, OWL only once, and stores them in memory, considering the resource limitation.

In addition to the efficient recourse handling, we use the Python language for resolving the heavy set operations (comparing and inserting web-service parameter sets) which happens frequently during composition processes, because Python provides high-performance functions to deal with this issue.

Besides, our implementation is designed to use SAX rather than DOM when it parse XML files because SAX is more efficient than DOM and runs really fast at runtime, in the situation where no need exist to build up a tree of elements and attributes. Finally, our implementation encompasses a module specific to this challenge that is the test file parser with aims to read test XML files.

Another feature of our implementation is the extension for providing the machine interface through the web-service interface. For the web-service interface, we provide a WSDL file which contains specification for operations, through which remote machines are allowed to access our core implementation via the Web. In addition to the machine interface, we also provide a java servlet page (JSP) for human users. Both interfaces are installed on a web application server, which deliver requests to our core algorithms such as web-service composition and discovery.

5. Related work and future directions

Several latest works are found in the web-service composition related literature. Preist et al. [4] presented a demonstrator system which applies semantic web-services technology to B2B integration, focusing specifically on a logistics supply chain. Their demonstration system is able to cope with all stages of the service life cycle – discovery, service selection and service execution. The proposed demonstrator system allows a requestor to discover logistics service providers, select appropriate logistics services, coordinate the services to form a composite service chain, and communicate with the service providers using arbitrary protocols through dynamic mediation.

Michalowski et al. [5] presented a running application, titled in Building Finder that integrates satellite imagery, geospatial data, and structured and semi-structured data from various online data sources using semantic-web technologies. Users can query an integrated view of these sources and request Building Finder to accurately superimpose buildings and streets obtained from various sources on satellite imagery. Building Finder is a real example that promises seamless integration of heterogeneous data from distributed sources, letting agents perform sophisticated and detailed data analysis. We plan to extend our approach by encompassing agent and semantic query techniques presented in [4] and [5], so that our system can deal with anonymous business partners on the fly, and search for the multiple heterogeneous distributed service information to accomplish the concept like dynamic supply chain.

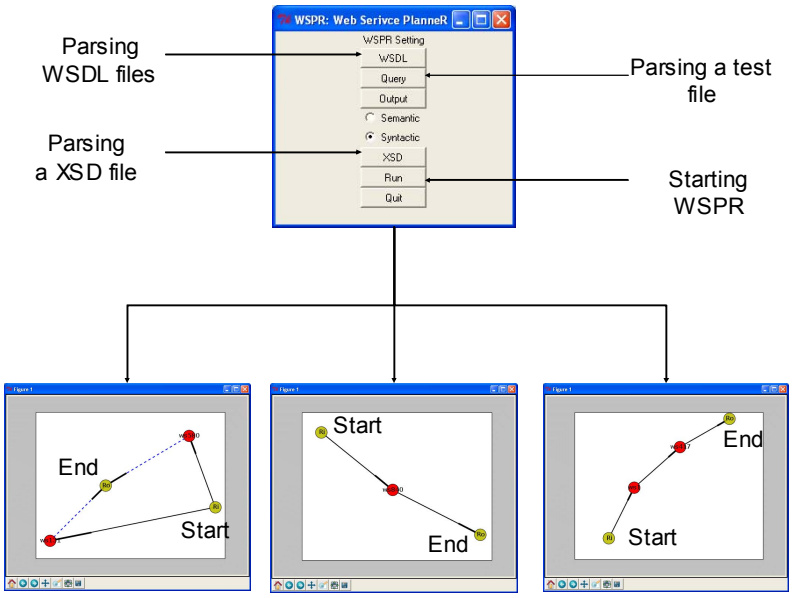

Figure 4. GUI of WSPR

6. Conclusion

A semantic web-service composition algorithm is presented to address the situation when the flexible parameter-matching framework is required. In practice, majority of public web services do not have annotated semantics yet, so that this flexible parameter-matching framework is evidently useful. In the future work, we will extend the current algorithm to adopt more complex methods (e.g., tree isomorphism, tree mapping, and approximate tree matching using Edit-distance).

7. References

[1] IEEE Joint Conference on E-commerce Technology (CEC'07) and Enterprise Computing, E-Commerce and E-Services (EEE'07). http://www.ws-challenge.org/wsc07/

[2] S. J. Russell and P. Norvig. "*Artificial Intelligence: A modern approach"*, Prentice-Hall, New Jersey, USA, 2002.

[3] S.-C. Oh, D. Lee and S. Kumara. "Web Service Planner (WSRP): An Effective and Scalable Web Service Composition Algorithm", *International Journal of Web Services Research*, 4(1), pp.1-22, 2007.

[4] C. Preist, J. Esplugas-Cuadrado, S. Battle, S. Grimm, and S. Williams. "Automated business-to-business integration of a logistics supply chain using semantic web services technology", *Proceedings of 4th International Semantic Web Conference*, pp. 987-1001, 2005.

[5] M. Michalowski, J.L. Ambite, S. Thakkar, R. Tuchinda, C.A. and Knoblock, S. Minton. "Retrieving and semantically integrating heterogeneous data from the web", *IEEE Intelligent Systems*, 19(3), pp. 72–79, 2006.