SWSDS: Quick Web Service Discovery and Composition in SEWSIP

Bin Xu, Tao Li, Zhifeng Gu, Gang Wu

Department of Computer Science and Technology, Tsinghua University, Beijing, P.R.China

{xubin,litao,gzf,wug }@keg.cs.tsinghua.edu.cn

Abstract

This paper is concerned with the efficiency of Web service discovery and composition. We proposed a quick Web service discovery and composition approach in SWSDS. In our approach, an index-based algorithm is adopted in both syntactic and semantic match during service discovery and composition. This paper is to introduce SWSDS and its discovery and composition algorithm.

1. Introduction

Web services are considered as self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Nowadays, many companies and organizations implement their core business and application services over Internet. Thus, the ability to efficiently and effectively select and integrate inter-organizational and heterogeneous services on the Web at runtime is an important step towards the development of the Web service applications.

There are many issues concerned with web services. In this paper, we especially focus on the efficiency of web services discovery and composition. By web service discovery, we mean a process of finding a web service that can satisfy user's request. By web service composition, we mean a process of combing existing services together in order to fulfill the user's request when no single web service can satisfy it.

We proposed a quick Web service discovery and composition approach in SWSDS (SEWSIP Web Services Discovery System). SEWSIP (SEmantic Web Services Integration Platform) is a platform which aims at service discovery, evaluation, selection, execution and semi-automatic composition [1].

In SWSDS, as illustrated in figure 1, the WSDL crawler crawls WSDL files directly from Internet[2]. WSDL files are annotated and converted to semantic description in OWL-S[3]. And finally, indexing is created based on WSDL and OWL-S files. Once a request is gained from user interface, the services that passed syntactic match and semantic match are returned to the requestor.

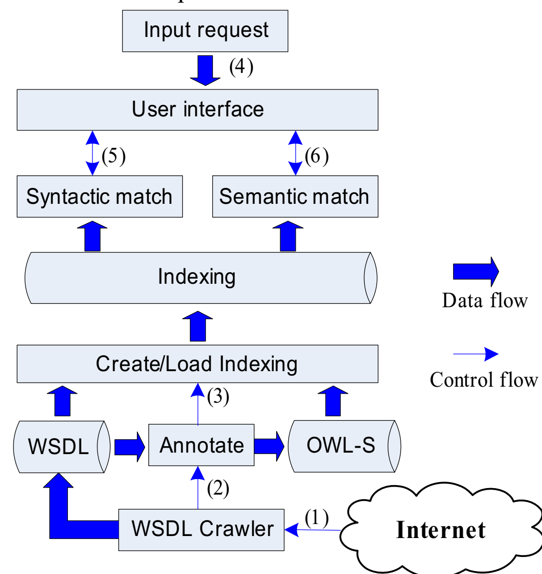

Figure 1. Control flow of SWSDS

2. Service Discovery and Composition

2.1 Definitions

In WSDL, there are many operations in one service. If we use an operation to represent a Web service, a Web service can be defined as following:

WS={IN, OUT}

IN ={iw_i | iw_i is operation's input data}

OUT={ow_i | ow_i is operation's output data}

Accordingly, formal description of request is:

$Request$={R_{in} , R_{out}}

R_{in}={ir_i | ir_i is input data given by requestor}

R_{out}={or_i | or_i is output data desired by requestor}

The formal description of a discovery task or composition task is to find services that:

IN R_{in} and OUT R_{out}.

2.2 Indexing

Proceedings of the 8th IEEE International Conference on E-Commerce Technology and the 3rd IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services (CEC/EEE'06)

0-7695-2511-3/06 $20.00 © 2006 IEEE

IEEE COMPUTER SOCIETY

We use inverted tables to build syntactic index for syntactic match, and semantic index for semantic match. The inverted table consists of two elements: the output ‘vocabulary’ and ‘occurrences’. The output ‘vocabulary’ is all output data appearing in all WSDL files. The output data are different in syntactic index and semantic index. In syntactic index, output data is ‘name’ of ‘part’ in ‘output message’ of WSDL. In semantic index, output data is ‘type’ of ‘part’ in ‘output message’ of WSDL.

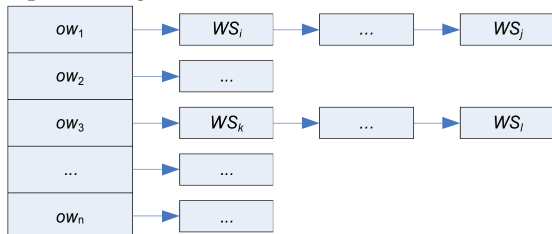

Figure 2. The inverted table for indexing

In syntactic index, for each output data, a list of services having this output data is called ‘occurrences’ of this output data. We assign a unique ID for each output data and each WSDL. Thus, the ‘vocabulary’ is actually an ID list. So is ‘occurrences’.

There is a little change about ‘occurrences’ in semantic index. Since the output data have hierarchy relations, the ‘occurrences’ of one output data must contain all of its descendants’ ‘occurrences’. The descendants are able to cover current ‘vocabulary’ in terms of the elements defined in Schema. For example, if “Ford” is defined as a subclass of “Car” and “Sail” is subclass of “Ford”, then ‘occurrences’ of “Car” should contain ‘occurrences’ of “Ford” and “Sail”.

By using index, it is easy to find services having ow_i and ow_j as its output data. It only needs to get the intersection of the ‘occurrences’ of ow_i and ow_j.

2.3 Discovery Algorithm

Discovery algorithm aims to discover single Web service that satisfies the request. There are three steps in discovery algorithm. The first step is to build index from the repository of WSDL files. The second step is to find candidate services which satisfy the output request. The third step is to find matched services from candidate services to satisfy the request input.

```
Input: A user request: <inputs, outputs>;
Output:  web services list: <matchedList>;
Algorithm: Discovery
//step1: create index
If (index exists)
     index = loadIndex();
     Else index = createIndex();
//step2: satisfy output request
candidateServices=null;
For each output o in outputs {
serviceList = Index.find(o); //find occurrences
If (candidateServices == null)
     candidateServices =serviceList;
     Else  candidateServices=candidateServices ∩
serviceList;
}
//step3: satisfy input request
result=null;
For each service s in candidateServices
if (s.inputs     inputs)
result.add(s);
return result;
```

2.4 Composition Algorithm

A composition task is to find a chain of Web services. In the chain illustrated in figure 3, one service can satisfy the inputs of its succeeding service either by its own outputs or the inputs of request. The inputs of the first service in the chain must be satisfied by the inputs of the request, and outputs of last service in the chain must satisfy the outputs of the request. The services between the first and last one are middle services.

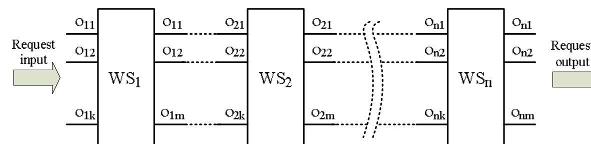

Figure 3. Web service chain

We conduct the composition search in a bottom-up mode. We firstly find the last service WS_n of a chain, whose output can satisfy the request output. Secondly we aim to find service WS_{n-1} that can satisfy the input of WS_n. As illustrated in figure4, we notice that the request input can also act as input of WS_n, outputs of WS_{n-1} need only to satisfy the rest inputs of WS_n. So we use the inputs of WS_n minus request input as a new output request to find previous service of WS_n -- WS_{n-1}. In this way we can get all the middle services. This searching stops when it can’t find any middle service. If the inputs of the last-found middle service can satisfy the request input, we get the right chain.

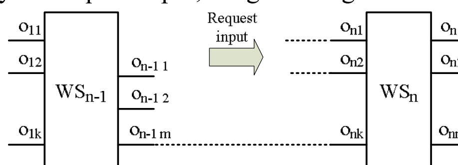

Figure 4. Middle service of a chain

```
Input: A user request: <inputs, outputs>;
Output: A collection of service composition;
Algorithm: Composition
```

Proceedings of the 8th IEEE International Conference on E-Commerce Technology and the 3rd IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services (CEC/EEE’06)
0-7695-2511-3/06 $20.00 © 2006 IEEE

IEEE COMPUTER SOCIETY

```
If (index exists)
index = loadIndex();
Else index = createIndex();
completed= false; // the flag to control the loop
For each s in S    // S is the set of all services
        s.isVisited=false;  // to avoid unending loop.
serviceList1 = getServicesByOutputs(outputs, inputs) //find services with outputs
While (!completed) {
completed = true;
serviceList2 = {};
For each s in serviceList1 {
If (!s.isVisited) {
s.isVisited = true; //service s is used in composition
 completed = false; /* some results are founded, so it's not the right time to quit. */
tmpList = getServicesByOutputs(s.inputs, inputs);
// let the inputs of 's' to be new outputs
If (tmpList != null) {
serviceList2 = serviceList2 ∪ tmpList;
For each ss in tmpList  ss.successor.add(s.id);
/* record the possible successor of web service 'ss' in the chain which is found here */
}}}
serviceList1 = serviceList2; //for next loop
} //end of While
For each s in S { // S is the set of all services
If (s.isVisted && (s.inputs     inputs)){// the head of one linked path for composition is found.
print(s.id);
ts=s;
/* find the composite path headed by s */
While (ts!=null) {
Print(ts.successor.id);
ts=ts.successor( );
}}} // end of composition
```

The method "getServicesByOutputs" will find all the services whose outputs can cover the required` outputs with the help of pre-provided inputs defined in the request.

```
List getServicesByOutputs(outputs, inputs){
candidateServices =null;
for each output o in (outputs-inputs) {/*use the output parts that are not included in the request inputs */
serviceList = Index.find(o);
If (candidateServices == null)
candidateServices =serviceList;
Else candidateServices = candidateServices ∩ serviceList; }
}//end of for
return candidateServices;
}
```

3. Discussion

There is another approach using index in service discovery and composition[4]. The difference is that we put both semantic and syntactic information in index. Current semantics of Web service is represented by the extension relation between different XML Schema types. It is just like another kind of syntactic with a little bit more complexity. Then we can put these information in index, and use the same discovery and composition algorithm. We found that in our experiment over 95% of the time is consumed on loading WSDL files while building index. After adopting index, the time consumed on discovery and composition is much less.

4. Conclusion

In this paper, we have proposed an approach for efficiently performing web services discovery and composition. By making use of the indexing technology, we accelerated the access of web services that can improve the discovery and composition processing.

We note that indexing is very important for web services discovery and composition. In practical applications, requests for retrieving services might be frequent. Indexing can significantly reduce the response time.

As the future work, we plan to make further improvement on the effectiveness and efficiency of service discovery. We also want to apply the proposed method to applications of web services.

5. References

[1]J.Z. Li, B. Xu etc. SEWSIP： Semantic based Web services Integration in P2P. In Proc. of SOSE2005, Beijing, China, 2005.

[2]B Xu, Y Wang etc. Web Services Searching based on Domain Ontology. In Proc. of SOSE2005, Beijing, China, 2005.

[3]P Zhang, J Z Li etc. Web Service Annotation Using Ontology Mapping. In Proc. of SOSE2005, Beijing, China, 2005.

[4]Huang S, Wang X, Zhou A. Efficient Web Service Composition Based on Syntactical Matching. In Proc. of EEE'05. HongKong, China, 2005.

IEEE COMPUTER SOCIETY