A QoS-Driven Approach for Semantic Service Composition

Yixin Yan, Bin Xu, Zhifeng Gu, Sen Luo
Department of Computer Science and Technology
Tsinghua University
Beijing, 100084, China
{yanyx, xubin, gzf, luos}@keg.cs.tsinghua.edu.cn

Abstract—**Semantic information, which is well-regulated and easy to be retrieved, has greatly enriched the expressive ability of the Web. These advantages can be applied in Web Services to meet the increasing complexity of Web applications. In this paper, we propose a service composition approach. It combines the large-scaled Web Services and semantic information which is described in WSC'09. Besides, QoS has become a critical issue to evaluate the performance of Web applications. Being different from improving the QoS of single services, our approach focuses on the overall QoS of the service composition. The algorithm shows that the semantic information based and QoS driven approach improves the efficiency and QoS performance of service composition.**

Keywords-service composition; QoS; semantic information integration

I. INTRODUCTION

Semantic Web has become a hot research topic in the past few years. With structured/semi-structured semantic information, it helps to construct a more understandable Internet environment as well as an easier way for Human-Computer interaction. And one of the most important targets is to turn the semantics from implicit to explicit, e.g. in the form of ontology. So most researchers in this field focus on the database, data mining and information retrieval technologies based on the semantic Web.

On the other hand, Web Service has become a popular technology in system design and implementation along with the development of SOA. It changes the behaviors of the Web from providing static information to providing services. Therefore, Web Service should support the analyzing, processing and transferring of semantic information [1].

However, as the business requirements become more complex, a single service usually cannot satisfy them. But meanwhile, the number of Web Services is also rapidly increasing, so the composition of Web Services provides a way to solve the problem. In this paper, we introduce a method for Web Service composition which is different from traditional ways. Especially, our method focuses on the following three aspects.

- Large-scale Web Service composition. The biggest challenge of handling a large number of services is the efficiency, e.g. real-time service composition in online search systems.
- Semantic information integration. WSC'09 provides a formal description of semantic information using Web Ontology Language (OWL) where two relationships of data and concept are recorded, i.e. class and sub-class; instance and concept.
- The QoS of the service composition.

The quality of service (QoS) is involved as an index to evaluate the efficiency of composition result in WSC'09. In practice, QoS is always a critical challenge of Web Services. Actually, many researchers from both colleges and industries doubt the efficiency of SOAP and proposed many methods to improve quality of services. E.g. reduce the number of functions in a Web Service in order to release the resource which is occupied by the agent to the client in time; or optimize the XML format including the data compression algorithm, XML parser, XML tags, etc.

However, in the problem of service composition, the quality of composite result rather than the quality of single service seems to be more important. Because services, which act as elements in the composite system, must coordinate to finish a given task, so the QoS of the composite system is more relevant with the composition structure than the QoS of single service [2, 3]. E.g. when several services are invoked in parallel, the bottleneck of response time will be the slowest service. But when the invocations are in a sequence, every service will affect the whole QoS. In fact, WSC'09 defines two popular indexes to evaluate the QoS of the composition system: response time and throughput. In this paper, we will introduce how to balance the topological structure and QoS performance of service composition.

The rest of this paper is organized as follows. Section 2 describes the problem proposed in WSC'09 and an overview of our approach. Section 3 detailedly presents the algorithm in a three steps search. Section 4 introduces the related works about QoS driven Web Services technologies. Section 4 gives the conclusion and future works.

II. OVERVIEW

A. Problem Overview

In previous ws-challenge competitions, a Web Service is defined by its input and output data. Additionally, QoS is introduced into the service model this year. So, a Web Service can be represented as, Service = { D_{in}, D_{out}, R, T} where

D_{in} = { d_i | $d_i \in$ Input data of the service},

D_{out} = { d_i | $d_i \in$ Output data of the service},

R = Response time measured in milliseconds,

T = Throughput measured in invocations per second.

It is important to emphasize that the service defined above can represent not only a single Web Service, but also a composition of Web Services. So service is an extended concept in the problem. Besides, as semantic information of services is denoted as class and sub-class relationship between the I/O data types, { D_{in}, D_{out} } can be extended to { D'_{in}, D'_{out} } as follows,

D'_{in} = { $d_i | d_i \in$ Input data or the sub-classes extended},

D'_{out} = { $d_i | d_i \in$ Output data or its super-classes}.

A request can be defined with { R'_{in}, R'_{out} } which has the similar meaning with { D'_{in} , D'_{out} }. If a service is determined by a given request, it means $D'_{in} \subseteq R'_{in}$ and $D'_{out} \supseteq R'_{out}$.

In order to evaluate the QoS of service composition, response time (R) and throughput (T) is introduced into the service model. The QoS index of the Web Service is indicated in a WSLA file. Based on the BPEL described composition result, WSC'09 defines the evaluation rules: For sequence A which consists of A1, A2, …, An; flow B which consists of B1, B2, …, Bn; Case C which consists of C1, C2, …, Cn,

$$R(A) = \sum_{i=0}^{n} R(A_i) \qquad Eq1$$
$$R(B) = \max\{R(B_1), R(B_2), ..., R(B_n)\} \qquad Eq2$$
$$R(C) = \min\{R(C_1), R(C_2), ..., R(C_n)\} \qquad Eq3$$
$$T(A) = \min\{T(A_1), T(A_2), ..., T(A_n)\} \qquad Eq4$$
$$T(B) = \min\{T(B_1), T(B_2), ..., T(B_n)\} \qquad Eq5$$
$$T(C) = \max\{T(C_1), T(C_2), ..., T(C_n)\} \qquad Eq6$$

Given all the available Web Services and their QoS index, the target is to find a service composition to satisfy a certain request, at the same time, with the lowest response time and highest throughput.

B. Approach Overview

Based on the request defined in the last section, we proposed a QoS-driven approach for semantic service composition. It contains three steps, shown in Figure 1, which ensures a result with good QoS performance can be found in a low cost of time.

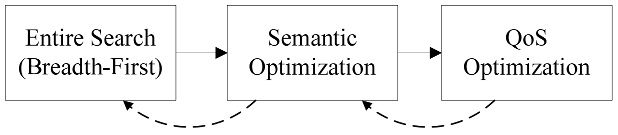

Figure 1. algorithm overview: 3 steps search

Generally speaking, the composition result can be regarded as a sequence with complex inner structure. According to the QoS evaluation rules Eq1 and Eq2, it is obvious that the longer the sequence is, the longer the response time will be. In contrast, if a sequence path is short, it means more parallel invocations exist in the solution. The same explanation can be applied in another index of QoS—throughput using Eq4 and Eq5. So the first step of our approach is a breadth-first search which finds the shortest path in global to satisfy the request.

Furthermore, when comparing the QoS of two Flows, the one which contains fewer invocations is more likely to have lower response time according to Eq2. Thus, in the second step, i.e. semantic optimization, the semantic information helps to cut redundant services so as to further minimize the response time. At the same time, the throughput will be enlarged according to Eq5.

Actually, the previous two steps ignore the QoS differences among different Web Services, but stand on the approximate estimation of QoS. The motivation of doing that is to reduce the search space of the third step, i.e. QoS optimization, and get a higher efficiency of the algorithm. So in the third step, we use the QoS index of the Web Services which is indicated in the WSLA file to refine the search result. Obviously, this approach cannot guarantee that the solution with the best QoS performance can be found. So when there is a solution with better QoS index outside the space determined by the first two steps, an automatic backtracking process will take place to enlarge the search space until the solution or a better one is found (shown by the broken arrows in Figure 1).

According to our preliminary experiments, the three steps algorithm ensures both the accuracy and efficiency of the search process.

III. QoS-Driven Semantic Service Composition

This section presents how our approach ensures the efficiency and effectiveness in a three steps search on the problem of service composition in WSC'09.

In fact, considering the entire efficiency, a pre-processing of data is done in the very beginning [4]. In the composition algorithm, we build an inverted table as the index for the data. And three kinds of relationship are recorded in it: 1. Relationships between data which are described in OWL; 2. Relationships between data and Web Services which are described in OWL; 3. QoS of Web Services which is described in WSLA. The index accelerates the composition process.

According to the semantic feature of data and the QoS evaluation baseline, we divide the composition task into three steps: 1. Top-down and breadth-first searching in the whole space; 2. Bottom-up optimization based on the semantic information; 3. QoS optimization based on step 2 and each Web Service's QoS index.

The problem of service composition can be regarded as a typical search problem on AND-OR graph [5]. Based on the service dependency graph (SDG), different optimization targets can be achieved through different strategies. A typical algorithm for AND-OR graph searching is AO*. However, AO* has two shortcomings: 1. AO* can only find one solution while we have two optimization targets, i.e. response time and throughput. Moreover, these two targets may conflict with each other. 2. The search speed is slow using AO*. A large-scale data set usually leads to a long sequence in the solution. So if we directly use the heuristic search, there will be many judgments and jumps inside the search process which make the efficiency low.

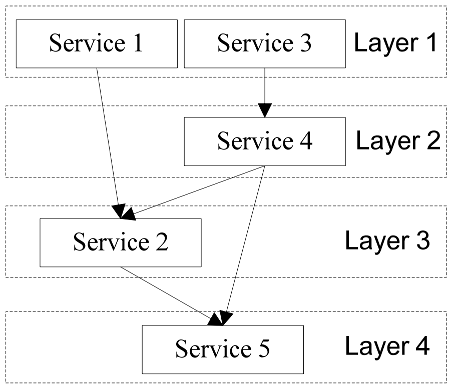

Figure 2. two expressions for a service composition: graph and sequence

Therefore, directly applying AO* to our problem cannot ensure the efficiency and the effectiveness. So we adopt a top-down and breadth-first search algorithm. A solution is represented by a sequence of layers instead of an AND-OR graph. See Figure 2.

To prove the two expressions logically equal to each other, we define a relationship between Web Services as {P, S} pairs where P represents the predecessor of S. A {P, S} pair means Web Service P directly invokes Web Service S. Accordingly, we define Set(P) which contains all P in all the {P, S} pairs. Circularly scan Set(P), for any Web Service W in Set(P), if W has not predecessor or all its predecessors have been marked as 'true', then W is marked as 'true'. Until all Web Services in Set(P) has been marked 'true', the sequence will be valid solution. E.g. we can get 5 {P, S} pairs: {S2, S5}, {S4, S2}, {S4, S5}, {S1, S2}, {S3, S4}. Figure 3 shows the process how to find an execution sequence.

Based on the above proof, our algorithm ensures the correctness of the result. Besides, the length of the result sequence will be the shortest. Starting with the provided data of the request, the algorithm finds all the covered Web Services, adds them into the first layer and then adds the output data of them into an available data set. Continuing the process until all the required data of the request is covered. See Listing 1.

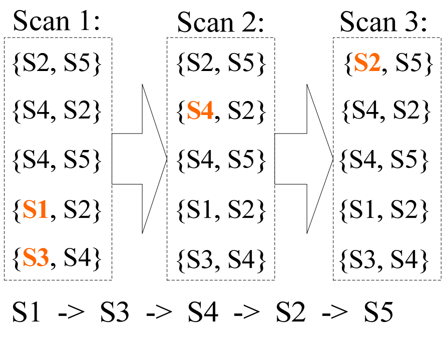

Figure 3. find a sequence equivalent to the graph

Listing 1: Step 1 — Bread-first entire search

```
while (!all request data are available) {
    //get a new layer
    Layer new_layer = dig();
    add_to_solution(new_layer);
    //check if the solution satisfies the request
    if(cover_quest())
        break;
    if(!solution_exist())
        exit(1);
} //end loop: solution found

//function: search in a new depth
Layer dig() {
    Layer new_layer;
    //search availble services
    foreach unused ws {
        if(is_ws_available(ws))
            new_layer.add(ws);
    }
    return new_layer;
} //end function
```

According to the algorithm, for any a Web Service in the solution, assume it belongs to the kth layer, then it can not belongs to the first k-1 layers in other solutions. That is because every layer in the solution has covered all possible invocations. The proof in details is ignored because of the limitation of pages. As mentioned in section 2, given the other conditions constant, the shorter the solution is, the better the paralleled feature will be.

The optimization in the second step focuses on how to decrease the number of services. Listing 2 shows how semantic relationship between data is used to cut redundant Web Services. First, define a predictive minimum data set for the solution and initial it with the required data. Then, search from bottom to up, if a data type can be provided by several services, select one of them and add its inputs into the minimum data set. Same process continues until the top layer is finished. After the semantic optimization, lots of redundant services are eliminated so as to ensure the QoS in every layer.

Listing 2: Step 2&3 — Semantic & QoS Optimization

```
1.  Solution optimization() {
2.    //record the global min set of available data
3.    Map min_data;
4.    //initial min_data: key<-required data; value<-false
5.    init(min_data);
6.    foreach layer in the solution {
7.      //record the refined layer
8.      Layer min_service;
9.      while ((data = min_data.next) != null) {
10.         //indicate the data has been handled
11.         data.value = true;
13.         //covered by the request data
14.         if(covered(data))
15.           continue;
16.         //get a service with the best QoS in current layer
17.         Service s = get_best_service(data, layer);
18          min_service.add(s);
19.         //enlarge the min_data with the data in
20.         //the input list of the service
21.         min_data.add(s.input_list);
22.     } //end loop: current layer refinement finish
23.     //update the current layer with fewer services
24.     updata_layer(layer, min_service);
25.   } //end loop:solution refinement finish
26. } //end function
```

The third step, i.e. the QoS optimization, focuses on the QoS index of each different Web Services. The 17th line in Listing 2 indicates the process of QoS optimization. In the search algorithm, the best service is selected from many candidate Web Services according to the QoS. However, a good QoS of current layer does not always leads to a good QoS of the entire solution. So we need to use backtracking until the best or a relative good solution is found. Furthermore, the backtracking may return to the first step, i.e. breadth-first search, in order to get a larger solution space.

Because response time and throughput are two different and irrelevant QoS indexes. It is hard to optimize both of them at the same time. So in our approach, they are separately considered using different search process.

IV. Related Works

Zeng [6] proposed a global planning approach for service composition to optimize multiple criteria of QoS. Through linear programming methods, it can handle multiple execution paths. Zeng's method focuses on the dynamic selection of Web Services in the runtime while our approach is static selection based on the data set.

Because of the disadvantages of UDDI (e.g. nearly half of the entries in UDDI are unavailable), Ran suggests extending the model of Web Service in [7]. Through adding the role of QoS certifier in service level, it ensures the QoS requirements of the service provider.

V. Conclusion and Future Works

We propose a service composition approach which makes use of the QoS and semantic relationship of Web Services. By this approach, a service composition with good QoS performance can be found through layered and backtracking search. Besides, the future improvement can focus on the I/O number of services. If a service requires many inputs, it seems to be harder to satisfy. In contrast, if a service can provide lots of data, it will easily have lots of successors. Another concern is the semantic complexity of data and concepts. If a concept acts as many others' father node, it is easy to get; otherwise, it will be hard to be mapped to other data types.

Acknowledgment

This work is supported by China National High-Tech Project (863) under grant No. 2009AA01Z120 and No. 2007AA010306.

References

[1] E Sirin, James Hendler Semi-automatic Composition of Web Services using Semantic Descriptions. Modeling, Architecture and Infrastructure workshop in ICEIS 2003.

[2] L Zeng, Benatallah, B. QoS-aware middleware for Web Services composition. IEEE Transactions on Software Engineering. Volume 30, 2004

[3] Gerardo Canfora. An approach for QoS-aware service composition based on genetic algorithms. The conference on Genetic and evolutionary computation.2005

[4] Zhifeng Gu, Bin Xu and Juanzi Li. Inheritance-Aware Document-Driven Service Composition. CEC/EEE'07, page 513, Tokyo, Japan, 2007. IEEE Computer Society.

[5] Y Yan, B Xu and Z Gu. Automatic Service Composition using AND/OR Graph. CEC/EEE'08, page 335, Washington D.C. 2008. IEEE Computer Society.

[6] L Zeng, B Benatallah. QoS-Aware Middleware for Web Service Composition. IEEE Transactions on Software Engineering, Vol. 30, No. 5, May 2004.

[7] Ran, Shuping: A Model for Web Services Discovery With QoS. ACM 2003.