Automatic Service Composition Using AND/OR Graph

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

Abstract

As SOC and Web service technology become more widely used, large amounts of services need to be efficiently and effectively composed to meet complex businesses. In this paper, we proposed an approach to resolve the composition problem over large-scale services. We used an inverted table as index for a quick service discovery, and applied a Service Dependency Graph (SDG) and an AND/OR graph as the algorithm basis for parallel compostion. Considering the semantic information described in Web service, our approach also recognizes and transmits the semantic relationships described in Web Ontology Language (OWL).

1. Introduction

Service Oriented Computing (SOC) is changing the way of conventional software designing and realization. And Web services technology, which is self describing, cross platform and loose coupling, is the basis to SOC for both its realization and popularization. Today, Web services over the Internet are used more frequently in real life, and the amount of them is growing rapidly, e.g. Amazon and ebay provide online product information to their potential consumers through Web services.

As more businesses could be executed through Web services, the requirement of Web service discovery and automatic service composition become larger. Service discovery and composition have become the most improtant research topics in SOC. Especially for large-scale Web services, the research on effective composition method with high efficiency is a critical challenge.

On the other hand, semantic composition becomes the trend of the web technology development. Many ontologies have been defined, created and used in various research fields and applications. However, the existing Web Service Description Languages and service composition methods can not give semantic information a full support for recognizing and transmitting.

In this paper, we propose a service discovery and composition model based on the AND/OR graph to resolve the semantic composition problem. We focus on both the parallelizability discovery and the efficiency of the algorithm over large-scale Web services. Section 2 presents the related work of service composition. Section 3 introduces related concepts and definitions including service dependency graph (SDG). In section 4 we present our approach of service composition. At last, we give the conclusion and future work.

2. Related Work

Many works on service discovery and automatic composition have been done based on the I/O data and the semantic information of services.

Liang [1] proposed a semi-automated method for service composition. The main idea is to construct an AND/OR graph form a service dependency graph (SDG), applying a bottom-up search algorithm REV* to find a sub-graph for the solution. It is a effective method to find executions on parallel, but it didn't consider the scale of the services. Besides, there have been many research on AND/OR graph searching such as AO* algorithm (A. Martelli, 1978) [4] [5]. Gu [2] presented a fast service composition model using a inverted table to index the services. [2] also proposed a method to handle the semantic relationship between I/O data.

Based on the previous work mentioned above, this paper presents a continued and improved method for service composition. In addition, our new algorithm has the following improvement:

- Search for the solution with parallel execution. Compared to the linear composition algorithm applied to the prior competitions, we use a graph to represent the solution for specified request, which can ignore some constraints on I/O to obtain more effective composition with high-level parallel executions.
- Fully support to the recognization, conversion and usage of semantic information, inheritance relationships between concepts, relationships between instances and concepts, etc., which are described in an OWL format.
- New evaluation function for the solution is applied, helping to improve the composition efficiency and accuracy.

1530-1354/08 $25.00 © 2008 IEEE
DOI 10.1109/CEC/EEE.2008.45

3. Definitions

In this section, we introduce the concepts and definitions related to the composition algorithm which will be discussed in the next section. For the convenience of discussion and the understanding of the concepts, here we give three assumptions in the paper. 1) The terms of "Web service" and "service" have the same meaning. 2) A composed service is considered as a service, they have the same definition, see section 3.2. 3) A service has and only has one operation.

3.1. Service Model

The service composition task proposed by WS-Challenge is basically based on the input and output data of the given Web Services. That is, for a particular service, we consider it as a black box which receives an input message and generate an output message, ignoring the other information provided in the WSDL such as namespace, binding, etc. Thus, a service can be simply defined as an I/O pair:

Definition 1 Service = $\{ D_{in}, D_{out} \}$ where

$D_{in} = \{ d_i \mid d_i \in$ the input data of service$\}$,

$D_{out} = \{ d_i \mid d_i \in$ the output data of service$\}$.

Besides, WS-Challenge adopts the semantic concepts as an enhancement to the syntactic based composition. The extended semantic relationship, classes and subclasses, between messages are described in an OWL file. And a Web service with a semantic extension is so-called a "Semantic Web Service". For example, if a service takes a parameter which is an instance of class X, and X has a subclass Y indicated in the OWL, then it will be considered a valid case if an instance of class Y is taken as the parameter designed for X. So we extend the I/O pair to $\{ D'_{in}, D'_{out} \}$ relying on the semantic extension:

Definition 2 Service = $\{ D'_{in}, D'_{out} \}$ where

$D'_{in} = \{ d_i \mid d_i \in$ service input data or the sub-classes extended from the input data$\}$,

$D'_{out} = \{ d_i \mid d_i \in$ service output data or the super-classes got form the output data$\}$.

Similarly, a Request can be defined as:

Definition 3 Request = $\{ R'_{in}, R'_{out} \}$ where

$R'_{in} = \{ d_i \mid d_i \in$ provided data by the requestor or the sub-classes extended$\}$,

$R'_{out} = \{ d_i \mid d_i \in$ required data by the requestor or super-classes extended$\}$.

Accordingly, we give the below statement to determine whether a found service satisfies a given request:

Statement 1 A Service $\{ D'_{in}, D'_{out} \}$ satisfies a Request $\{ R'_{in}, R'_{out} \}$ iff they satisfy the restriction $D'_{in} \subseteq R'_{in}$ and $D'_{out} \supseteq R'_{out}$.

3.2. Service Dependency Graph

In Figure 1, service A and B are linked with directed arrows through a data node 'd4' which the former service generates and the later one consumes. We call two services which have this kind of relationship have dependency. And if all the given services are linked and represented using dependencies, we call it a services dependency graph (SDG).

Considering the semantic extension, we use broken line arrows to indicate the inheritance relationship between data nodes. In figure 1, 'd5' is a subclass of 'd6', so 'd5' can be taken as the parameter of service B.

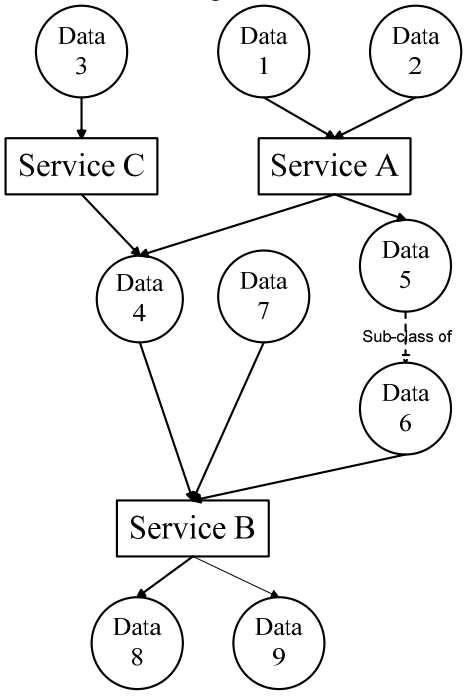

Fig.1 Example of Service Dependency Graph (SDG)

4. Service Composition Approach

In this section, we propose the search algorithm that finds a composed service with parallel optimization satisfying the given request. In section 4.1, we will introduce the further formalization of service dependency graph. In section 4.2, an efficient indexing method of large-scale Web services is described. Section 4.3 gives the core search algorithm.

4.1. Generate AND/OR Graph

In the service dependency graph (SDG), there are several obvious regulations. 1) Any path of the SDG consists of two kinds of nodes which always alternate in the path. 2) We call a service is satisfied when all the income lines (inputs) is satisfied. 3) In a found solution, it's not necessary to satisfy all the income lines of an OR data node. For example, 'd4' is used in path from Service A to B regardless of Service C in figure 1.

Based on the above features in the SDG, we can generate a specified AND/OR graph using the following rules respectively:

- Map a service node to an AND node.
- Map a data node to an OR node.
- Add a dummy AND node $\{D'_{in}, \text{null}\}$ to the graph to connect all the requested data nodes by the user ($D'_{in} = R'_{out}$).
- Collect all the data nodes provided by the user as the target set.

4.2. Indexing of Large-scale Services

WS-Challenge provides a large amount of candidate Web services. So it becomes very important to build a high performance index for service lookup. In our approach, the index of services is constructed as an inverted table. It can be explained as a map from the I/O data to the services which generate the data. Using the inverted table, it will return all the involved services through a given I/O data very quickly.

Besides, in order to meet the requirement of semantic extension, we consider the super class of an output data of a service as the output data of the service as well. Thus, it will not be necessary to consider the semantic relationship separately in the search algorithm.

4.3. Composition Algorithm

For a given request (defined in Definition 3), the task proposed by WS-Challenge is to find a composed service (defined in Definition 2) that satisfies the request (defined in Statement 1).

Before presenting the search algorithm, we first discuss the feasibility of the algorithm based on the AND/OR graph generated from the SDG described in section 4.1. In fact, it is easy to prove that a solution is a sub-graph of the AND/OR graph which starts from the dummy AND node $\{D'_{in}, \text{null}\}$ and ends at the target set R'_{in} presented in figure 2. Because all the required data nodes are connected to the dummy AND node ($D'_{in} = R'_{out}$), they are satisfied iff the starting node is satisfied ($D'_{out} \supseteq R'_{in} = R'_{out}$). On the other hand, we mean the sub-graph ends at the target set by $D'_{in} \subseteq R'_{in}$. Then we have proved the equivalence of the sub-graph and the solution service.

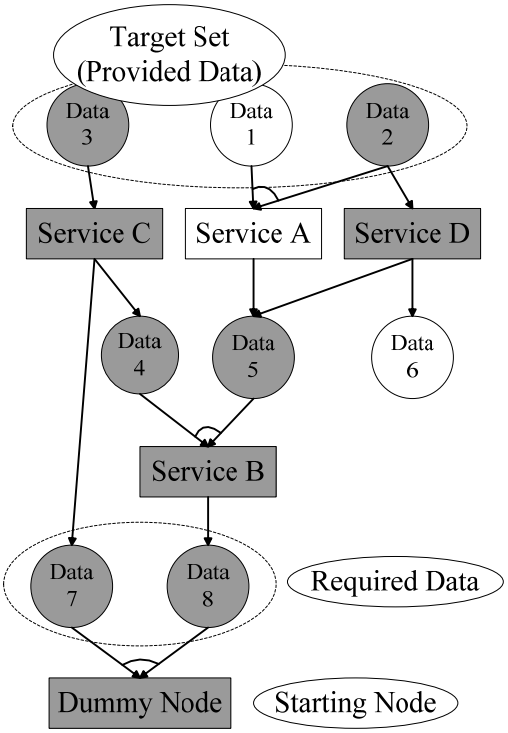

Fig.2 A solution from AND/OR graph

One of the aims of this paper is to find the optimal service composition solution from a given set of services. Commonly, the more services a solution calls, the more it will cost in time. And the more inputs a service takes, the harder it can be satisfied. So it would be reasonable to define a cost of a service and a cost of data in order to represent the cost of the solution. In this task, we simply assign a fixed cost c_s on each AND node except for the dummy starting node and another fixed cost c_d for each OR node except for the nodes in the target set. Using the cost values, our algorithm will find the smallest cost solution if existed for a given request.

First, we mark every element in the target set as 'RESOLVED' because these are the given inputs by the requestor, and other nodes are marked as 'UNRESOLVED'. Any AND node is modified as 'RESOLVED' iff every its input node is marked 'RESOLVED'. Any OR node will be modified as 'RESOLVED' iff its producer AND node is marked 'RESOLVED'. We define graph G for recording the nodes and the path of the solution. G contains the starting dummy node and an empty path initially. Actually G could be thought as a changing directed sub-graph of the AND/OR graph.

Algorithm for Service Composition

```
1.  //add(s, G), s is the starting node
2.  init_graph(G);
3.  Until s is marked "RESOLVED", begin
4.     //find a node to expand
5.     node n = find(G);
6.     //expand node
7.     expand(n, G);
8.     //modify cost, link and mark
9.     void modify(n, G);
10.    end begin//end loop

11. /* function: bottom-up modify costs, links and marks
12.    of the nodes recursively*/
13.  void modify(n, G){
14.     if n == s then
15.        return;
16.     end if
17.     modify_mark(n);
18.     modify_cost(n);
19.     modify_link(n);
20.     foreach parent p of n do
21.        modify(p);
22.     end for
23.  }//end modify
```

The algorithm starts from G. Repeatedly get an 'UNRESOLVED' node from G accroding to the current path pointers. Expand the node and add the new generated nodes to G. Modify the 'UNRESOLVED' mark if possible. Then modify the cost of all related node in the solution path recursively. We define the cost modification formulas as follows:

$$Cost(n) = \begin{cases} \sum_{input\ I_i} Cost(I_i), & n\ is\ AND\ node \\ \max_{producer\ P_i} \{Cost\{P_i\}\}, & n\ is\ OR\ node \end{cases}$$

When the cost modification is finished, redirect the current path pointer. Repeat the loop until all the nodes of G is marked as 'RESOLVED', otherwise, claim no solution. The sequential nodes directed by the path pointers in G is a solution with a minimum cost.

At the end, we represent the found solution to a process flow format such as a WSBPEL document defined by WS-Challege.

5. Conclusion, Analysis and Discussion

Our approach is based on an AND/OR graph, and the found solution is a directed sub-graph of the given services. Branches, which contain at least one AND node, flowing out from a node can be executed in parallel.

Another quest proposed is semantic composition. Our search model supports the discovery and transmitting of the ontology information described in an OWL format.

Considering the composition efficiency, we adopt a fast service discovery method using an inverted table mapping each attribute to its owner services.

However, there still have some problems and improvements to be realized before applying our approach to practical use:

- The functionalities of services are not considered in our composition method. Obviously, this will cause confusion when two services take and generate exactly the same types of data. To avoid the confusion, other information besides I/O data must be used. Actually, this problem is still a critical challenge.
- Parallel evaluation on a solution can be taken as the heuristic information in the algorithm. However, there's not a benchmark for evaluating a composed service and the composition itself that is widely accepted. For our algorithm, how to balance getting the best solution against getting all the solutions also depend on future experiments.

6. Acknowledgement

This work is supported by China National High-Tech Project (863) under grant No.2007 AA 010306 and China Postdoctoral Science Foundation under the grant No. 20070410061.

7. References

[1] Q. A. Liang and S. Y.W. Su. AND/OR graph and search algorithm for discovering composite web services. International Journal of Web Services Research, 2(4):48 – 67, 2005.

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

[3] M. Aiello, C. Platzer, F. Rosenberg, H. Tran, M. Vasko, and S. Dustdar. Web service indexing for efficient retrieval and composition. In CEC/EEE'06, page 63, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[4] Patrick Henry Winston. *Artificial Intelligence (3rd edition)*, Addison Wesley, page94, May 10, 1992.

[5] A. Martelli and U. Montanari, Optimizing decision trees through Heuristically guided search. Commun. ACM., vol. 21, no. 12, pp. 1025–1039, 1978.