ABSTRACT

In current service-oriented applications, each individual Web service usually cannot satisfy users’ requests for Web services, but the functionalities of different Web services may be integrated to accomplish requested task. This process is called Web service composition, and the resulted service is called composite Web service. The discovery and composition of Web services are two important research issues. In this article, a Web service composition system, BITS, which won the championship of the ICEBE05 contest due to its outstanding performance, is introduced. BITS is implemented based on a binary tree which is used to organize Web services. In this system, some search strategies and optimization techniques are developed for efficient service discovery and service composition. Extensive experiments are conducted for performance study, and the results show the efficiency and effectiveness of the methods adopted in BITS.

Keywords: service composition; service discovery; Web service

INTRODUCTION

Efficient service discovery and composition methods for service-oriented applications (SOA) (Yue et al., 2004) are both critical and challenging. Recently, various contests were held in conjunction with international conferences to promote the development of novel techniques and systems for efficient service discovery and service composition. These international conferences include 2005 IEEE International Conference on e-technology, e-commerce and e-service (EEE05 contest, 2005) and 2005 IEEE International Conference on e-Business Engineering (ICEBE05 contest, 2005), etc.

To better understand the services composition problem, we consider the following example. Table 1 describes a services repository, where \( ws \) represents a service with input parameter \( ws\text{.in} \) and service output \( ws\text{.out} \). The user’s request is expressed as \( R = (R\text{.i}, R\text{.o}) \), where \( R\text{.i} \) is the input given by the user, and \( R\text{.o} \) is the requested output. The procedure of finding an individual service \( ws \) for user request \( R \) such that \( R\text{.i} \supseteq ws\text{.in} \), and \( ws\text{.out} \supseteq R\text{.o} \) is called “service discovery.” For instance, if
there is a service discovery request $R_1$, where $R_1.i = \{\text{ZipCode, Road, District}\}$, $R_1.o = \{\text{Weather, Address}\}$, then $ws_1$ is the discovery result for this request, since $\{\text{ZipCode, Road, District}\} \supseteq ws_1.in$ and $ws_1.out \supseteq \{\text{Weather, Address}\}$.

If no single service can meet the requirement, two or more services are integrated to perform the requested task. This procedure is called service composition. For example, there is no single service in Table 1, which satisfies the request $R_2$, where $R_2.i = \{\text{Zone, Road}\}$, $R_2.o = \{\text{Weather, Address, Province}\}$. ($R_2$ will be used as the example in subsequent sections, but it is appearing as $R$). With the input parameters $\{\text{Zone, Road}\}$, $ws_2$ and $ws_3$ are to be invoked, but neither provides the required output. However, by taking the union of the outputs of $ws_2$ and $ws_3$, we have $(ws_2.out \cup ws_3.out) \supseteq R.o$. In this case, two services compliment one another to satisfy the requirements.

The primary goal of service composition is to find suitable services, then these services are scheduled to satisfy users’ request. The scheduling plan is called invocation pipeline. Figure 1 illustrates the invocation pipeline for request $R$. In Figure 1, every preceding node acts as one of the conditions of the succeeding node. In this example, the pre-condition for invoking $ws_j$ is a set of parameters, abbreviated to $ps$ (such that $ps \supseteq ws_j.in$), which can be obtained from previous invocations.

There exists some related work, such as UDDI (2002) from the industry and Woogle (Dong et al., 2004) from the academia. However, finding potential invocation pipelines as well as service compositions efficiently remains an open problem. The challenge lies in the fact that the number of candidate services can be very large. Our article focuses on addressing this problem by proposing some strategies to find candidate invocation pipelines efficiently. In this article, we propose a novel service discovery and service composition method to design a binary tree based Web service composition System, called BITS. BITS won the championship in the ICEBE 2005 Web services challenge contest (ICEBE05 Contest Result, 2005). The major contributions of this article are as follows:

1. A succinct binary tree approach is proposed, which enables service composition problem to be transformed into a tree representation and traversal problem.
2. An efficient strategy for finding pre-conditions for each internal node in the invocation trees is designed. Theoretical

<table>
<thead>
<tr>
<th>Web Service</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ws_1$</td>
<td>ZipCode, District</td>
<td>Weather, Address</td>
</tr>
<tr>
<td>$ws_2$</td>
<td>Zone</td>
<td>ZipCode, Province</td>
</tr>
<tr>
<td>$ws_3$</td>
<td>Road</td>
<td>Temperature, District, City</td>
</tr>
<tr>
<td>$ws_4$</td>
<td>ZipCode, City</td>
<td>Weather, Address, Temperature</td>
</tr>
<tr>
<td>$ws_5$</td>
<td>Longitude, Latitude</td>
<td>City</td>
</tr>
</tbody>
</table>

Table 1. Web services repository
Related Content

Privacy-Preserving Trust Establishment with Web Service Enhancements
www.igi-global.com/chapter/privacy-preserving-trust-establishment-web/59918?camid=4v1a

Proactively Composing Web Services as Tasks by Semantic Web Agents
www.igi-global.com/chapter/proactively-composing-web-services-tasks/26920?camid=4v1a

Modeling and Respecting Privacy Specification when Composing DaaS Services*
www.igi-global.com/article/modeling-and-respecting-privacy-specification-when-composing-daas-services/80177?camid=4v1a
The DeltaGrid Service Composition and Recovery Model
www.igi-global.com/article/deltagrid-service-composition-recovery-model/34105?camid=4v1a