ABSTRACT

As the emergence of service-oriented architecture provides a major boost for e-commerce agility, the number of available Web services is rapidly increasing. However, when there are a large number of Web services available and no single Web service satisfies the given request, one has to “compose” multiple Web services to fulfill the goal. In this article, toward this problem, we present an AI planning-based Web service composition algorithm named as WSPR. We evaluate the efficiency and effectiveness of WSPR using two publicly available test sets—EEE05 and ICEBE05. In addition, we analyze the two test sets and suggest several improvements to benchmark Web service composition better.

Keywords: AI planning; parameter usage distribution; Web services composition

INTRODUCTION

Web Services are often considered one of the most important and vital building blocks for the semantic Web (Berners-Lee, 2001). As such, the industrial support of Web services has grown drastically in recent years. For example, it is expected that by 2007, 72 percent of all application development software will support Web services and 45% of all types of software will be Web services enabled (Cantera, 2004). In Web services enabled networks, typically, a client program first locates a Web services server that can satisfy certain requests from a yellow page (UDDI), and obtain a detailed specification (WSDL) about the service. Then, using the known API in the specification, the client sends a request to the Web service considered via a standard message protocol (SOAP), and in return receives a response from the service. Unlike conventional programming interface, Web services are self-explanatory so that by interpreting XML tags, applications can interpret the semantics of operations. In particular, the problem of practical interest concerns the following two issues. Given a request $r$, among thousands of candidate Web services found in UDDI: (1) How to find matching services that satisfy $r$; (2) How to compose multiple services
to satisfy \( r \) when a matching service does not exist. We motivate our work through the following example.

**Motivating Example**

Consider the four Web services in Table 1 illustrates in WSDL notation:

- Given the hotel, city, and state information, findHotel returns the address and zip code of the hotel.
- Given the zip code and food preference, findRestaurant returns the name, phone number, and address of the restaurant with matching food preference and closest to the zip code.
- Given the current location and food preference, guideRestaurant returns the address of the closest restaurant and its rating.
- Given the start and destination addresses, findDirection returns detailed step-by-step driving direction and a map image of the destination address.

Now, consider the following two requests from “State College, PA, USA”:

1. \( r_1 \): find the address of the hotel *Atherton*, and
2. \( r_2 \): find a *Thai* restaurant near the hotel *Atherton* along with driving directions.

To fulfill \( r_1 \), invoking the Web service findHotel is sufficient. That is, by invoking findHotel(“Atherton,” “State College,” “PA”), one can get the address of the hotel as “100 Atherton Ave” with the zip code of “16801.” However, none of the four Web services can satisfy \( r_2 \) alone. Both Web services, findRestaurant and guideRestaurant, can find a *Thai* restaurant near the hotel, but cannot provide driving directions. On the other hand, the Web service findDirection can give driving directions from

<table>
<thead>
<tr>
<th>Table 1. Example Web services</th>
</tr>
</thead>
</table>

- **(a) findHotel**
  ```
  <message name="findHotel_Request">
    <part name="hotel" type="xs:string">
    <part name="city" type="xs:string">
    <part name="state" type="xs:string">
  </message>
  <message name="findHotel_Response">
    <part name="address" type="xs:string">
    <part name="zip" type="xs:string">
  </message>
  ```

- **(b) findRestaurant**
  ```
  <message name="findRestaurant_Request">
    <part name="zip" type="xs:string">
    <part name="foodPref" type="xs:string">
  </message>
  <message name="findRestaurant_Response">
    <part name="name" type="xs:string">
    <part name="phone" type="xs:string">
    <part name="address" type="xs:string">
  </message>
  ```

- **(c) guideRestaurant**
  ```
  <message name="guideRestaurant_Request">
    <part name="foodPref" type="xs:string">
    <part name="currAddress" type="xs:string">
  </message>
  <message name="guideRestaurant_Response">
    <part name="rating" type="xs:string">
    <part name="destAddress" type="xs:string">
  </message>
  ```

- **(d) findDirection**
  ```
  <message name="findDirection_Request">
    <part name="fromAddress" type="xs:string">
    <part name="toAddress" type="xs:string">
  </message>
  <message name="findDirection_Response">
    <part name="map" type="xs:string">
    <part name="direction" type="xs:string">
  ```

Copyright © 2007, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.
Dynamic, Flow Control-Based Information Management for Web Services
www.igi-global.com/chapter/dynamic-flow-control-based-information/31216?camid=4v1a

Automated Situation-Aware Service Composition in Service-Oriented Computing
www.igi-global.com/article/automated-situation-aware-service-composition/3109?camid=4v1a