Chapter 3.5
WSBen:
A Web Services Discovery and Composition Benchmark Toolkit[1]

Seog-Chan Oh
General Motors R&D Center, USA

Dongwon Lee
The Pennsylvania State University, USA

ABSTRACT

In this article, a novel benchmark toolkit, WSBen, for testing web services discovery and composition algorithms is presented. The WSBen includes: (1) a collection of synthetically generated web services files in WSDL format with diverse data and model characteristics; (2) queries for testing discovery and composition algorithms; (3) auxiliary files to do statistical analysis on the WSDL test sets; (4) converted WSDL test sets that conventional AI planners can read; and (5) a graphical interface to control all these behaviors. Users can fine-tune the generated WSDL test files by varying underlying network models. To illustrate the application of the WSBen, in addition, we present case studies from three domains: (1) web service composition; (2) AI planning; and (3) the laws of networks in Physics community. It is our hope that WSBen will provide useful insights in evaluating the performance of web services discovery and composition algorithms. The WSBen toolkit is available at: http://pike.psu.edu/sw/wsben/.

INTRODUCTION

A *Web Service* is a set of related functionalities that can be loosely coupled with other services programmatically through the Web. Examples of web applications using Web services include weather forecasting, credit check, and travel agency programs. As a growing number of Web services are available on the Web and in organizations, finding and composing the right set of Web services become ever more important. As a result, in recent years, a plethora of research work and products on Web-service discovery and composition problems have appeared[2]. In addition, the Web service research community

has hosted open competition programs (e.g., ICEBE05[3], EEE06[4]) to solicit algorithms and software to discover pertinent Web services and compose them to make value-added functionality. Despite all this attention, however, there have been very few test environments available for evaluating such algorithms and software. The lack of such a testing environment with flexible features hinders the development of new composition algorithms and validation of the proposed ones. Therefore, the need for a benchmark arises naturally to evaluate and compare algorithms and software for the Web-service discovery and composition problems. As desiderata for such a benchmark, it must have (a large number of) web services in the standard-based WSDL files and test queries that can represent diverse scenarios and situations that emphasize different aspects of various Web-service application domains. Often, however, test environments used in research and evaluation have skewed test cases that do not necessarily capture real scenarios. Consider the following example.

Example 1 (Motivating) *Let us use the following notations: A Web service $w \in W$, specified in a WSDL file, can be viewed as a collection of operations, each of which in turn consists of input and output parameters. When an operation op has input parameters $op^i = \{p_1,...,p_n\}$ and output parameters $op^o = \{q_1,...,q_n\}$, we denote the operation by $op(op^i, op^o)$. Furthermore, each parameter is viewed as a pair of (name, type). We denote the name and type of a parameter p by p.name and p.type, respectively. For the motivating observation, we first downloaded 1,544 raw WSDL files that Fan and Kambhampati (2005) gathered from real-world Web services registries such as XMethods or BindingPoint. We refer to the data set as PUB06. For the purpose of preprocessing PUB06, first, we conducted WSDL validation according to WSDL standard, where 874 invalid WSDL files are removed and 670 files are left out. Second, we removed 101 duplicated WSDL files at operation level, yielding 569 valid WSDL files. Finally, we conducted type flattening and data cleaning processes subsequently. The type flattening process is to extract atomic types from user-defined complex types using type hierarchy of XML schema. This process helps find more compatible parameter faster. Details are found in (Kil, Oh, & Lee, 2006). The final step is the data cleansing to improve the quality of parameters. For instance, substantial number of output parameters (16%) was named "return", "result", or "response" which is too ambiguous for clients. However, often, their more precise underline meaning can be derived from contexts. For instance, if the output parameter named "result" belongs to the operation named "getAddress'", then the "result" is in fact "Address". In addition, often, naming follows apparent pattern such as getFooFromBar or searchFooByBar. Therefore, to replace names of parameters or operations by more meaningful ones, we removed spam tokens like "get" or "by" as much as we could.*

We measured how many distinct parameters each WSDL file contained. Suppose that given a parameter $p \in P$, we denote the number of occurrences of p.name as #(p.name). That is, #("pwd") indicates the number of occurrences of the parameter with name of "pwd". Figure 1 illustrates #(p.name) distributions of PUB06 and the ICEBE05 test set, where the x-axis is #(p.name) and the y-axis is the number of parameters with the same #(p.name) value. The distribution of PUB06 has no humps. We also plotted a power-function, over the #(p.name) distribution, and found that the exponent is 1.1394. Although 1.1394 does not suffice the requirement to be the power law (Denning, 2004), the distribution is skewed enough to be seen as the Zipf-like distribution. Indeed, the parameters such as "license key", "start date", "end date," or "password" have a large #(p.name) value, while most parameters appear just once. This observation also implies the existence of hub parameters, which appear in Web services frequently, and serve important roles