Web Service Composition: a Reality Check

[1] Jianguo Lu, [2] Yijun Yu, [1] Debashis Roy, [1] Deepa Saha

[1]School of Computer Science, University of Windsor
{jlu, roy17, sahag@cs.uwindsor.ca

[2] Computing Department, The Open University
y.yu@open.ac.uk

Abstract

Automated web service composition is one of the major promises of service-oriented architecture, where services can be discovered and composed dynamically and automatically. To investigate the methods for composite web service construction, we conducted an experiment on creating useful composite web services from real existing web services where semantic annotations are not available. The empirical study reveals the difficulties and research challenges in the discovery, invocation, and composition of web services. The automation of web service composition requires the inputs from both services providers and service consumers. Service providers need to develop high quality services in a disciplined and collaborative way, and service consumers need to be equipped with tools providing helps such as service discovery and matching.

Keywords: Web service discovery, web service composition, empirical study.

1. Introduction

Web service is designed to be reused and composed with other web services, manually or automatically. The ultimate goal of service oriented architecture is the automated discovery and automated composition of web services. There have been substantial researches on service discovery [2] [17] and composition [4] [16], based on various formal methods and AI technologies [3] [8] [9] [11] [12]. In the mean while, on the web there are already tens of thousands useful web services that are accessible to the public [5] [21]. However, most of the research and the resulting prototypes target on imaginary web services, usually with semantic annotation, instead of real ones. As a result, there are few tools available that assist the creation of real composite web services from existing publicly available one.

To identify the research challenges in the whole process of web service discovery, invocation, and composition, we conducted an experiment involving 23 graduate students. They are requested to create novel and useful web services out of existing ones on the web, and report their experience on web service discovery, invocation, and composition. The purpose of the experiment is to identify and evaluate the existing methods and tools that can be used in real web service discovery and composition, and identify the difficulties and research problems in real web service composition.

The study shows that the construction of composite web service is a difficult task that requires creativity. The most difficult part in developing new web services is the discovery of the pertinent web services to achieve the goal. Due to various restrictions in existing web services, currently it is almost impossible to have automated web service composition. What is even worse is that manual composition is much more difficult than writing a conventional program because of the ad hoc nature of existing web services which lacks disciplined development and maintenance.

2. The Experiment

23 graduate students are formed into 11 groups. Many of them have excellent programming skills and IT industry experience. Each group is required to produce one or more novel and useful composite service from existing ones. Table 1 lists the composite services they generated. Please notice that although only 12 composite services are generated, many compositions consist of several atomic services. Before a successful composition scenario and atomic service can be decided, students have to investigate many existing services. Hence a large number of web services are tackled.

Before the experiment, students read extensively on both practical and theoretical aspects of web service and semantic web, and are exposed to a variety of web service discovery and composition methods.

In the process the students record the difficulties they encounter in service discovery, invocation, and composition.

Table 1: List of Composite Web Services

	Composite service(s)	**Component web services**
1	Truck driver route planning and notification	Mappoint, across communications, fast weather
2	Medicare map	Medicare, google map
3	Price comparison and conversion	Amazon, Barnsnoble, currency converter
4	Trip map	Yahoo trip, google map
5	e-commerce product rating	Amazon, google, msn search, currency converter
6	Price comparison	Ebay, amazon, currency converter
7	Geo tag	Flicker, google map, weather
8	Nearest airport	Airport, distance, Zipcode
9	Encrypted email	Encryption, email
10	Composite calendar	Calendar, reminder, call
11	Airport weather	Airport, google map, weather
12	Phone weather	Cell phone text message, weather

3. Web service discovery

The first step in web service composition is to locate the pertinent web services that can be used in our composition task. There has been substantial research in web service discovery and the UDDI [2] [18] [22]. In practice, students prefer two approaches to discovering web services manually, i.e., from web service portals, or from generic search engines such as google.

Most students used web service portals such as XMethods.com to locate web services, although they are familiar with researches in web service discovery. None of the students tried to use web service discovery tools or prototypes reported in literature. The reasons may be that most web service discovery researches focus on the semantic annotation of web services and semantic and capability matching of web services. However, existing web services on the web usually are not equipped with semantic descriptions or capability specifications.

Although the number of web services in each portal is limited (a few hundreds in general), most of those web services are valid or active ones. Two groups of students used programmable web APIs [13] instead of WSDLs, from which they generated more complex and interesting applications. Strictly speaking, some APIs from ProgrammableWeb are not the traditional web services. For example, they may not provide wsdl descriptions.

Here are the methods students use to find web services. Many groups use a combination of different methods. For example, they may search for google first to have a rough idea whether there are certain type of services, then go to a service portal to locate the ones that are active.

Table 2: Places to find web services

Portals to find web services	Number of groups used the portal
Google.com	3
Xmethods.com	5
ProgrammableWeb[13]	3
WebserviceList.com	2
WebserviceX.com	4
strikeIron.com	1
Trynt.com	1
Wsindex.com	1
remoteMethods.com	1

Given the fact that none of the web service portals contains large number of web services, and yet they are still the most popular ways to discover pertinent services, there is a need to build a larger web service repository where the quality of web service is put in the first priority.

4. Web service invocation

Once a web service is located, we need to call the web service. To automate the composition process, first web services have to be invoked automatically.

In theory, given the WSDL description of a web service, there is no problem to invoke it automatically. WSDL is designed to enable automated invocation. Indeed there are various tools supporting this in various programming languages. For the example of Java programming, Apache Axis can generate the corresponding support Java classes to call the web services with minimal programming effort.

During the experiment, there are a few factors that make the automated invocation impossible, including registration key and soap head change. In fact, many times students found that even the manual invocation is difficult due to lack of documentation and quick evolution of web services.

4.1 Difficulties for automated invocation

4.1.1 Registration key

In the experiment we find that the current web services are not easy to invoke even when the WSDLs are available and the services are active. The main reason is that most web services require a registration key that has to be obtained manually. Almost all the web services that are investigated in our experiment need manual registration. Those service providers require users fill in registration forms manually, so that we can receive the registration keys by email. Then the key has to be provided as a parameter in the operation each time the web service is called. In addition to manual registration, some services even require payments. This requirement from service providers practically prohibits any automated web service invocation.

4.1.2 Tools may fail

Except the registration key issue, most web services can be invoked by using tools such as Apache Axis. Basically, given the WSDL as input, Axis can generate a set of java classes corresponding to the Schemas in the WSDL description, and other classes supporting the remote call. Then the invocation of web service is simply to initiate some classes that are generated by the tool.

However, sometimes web services require something that can't be generated by the tools.

Example 1: *Geoplaces* web service requires the authentication ID to be sent in SOAP header instead of in a parameter of the operation. But WSDL2Java in Axis does not generate the corresponding java code filling in the SOAP head properly. As a result we had to manually create the SOAP header element and attach it to the stub object before calling any function.

The format of the SOAP request for the *GetPlaceDetails* operation in *Geoplaces* is given below. The italic part has to be manually injected.

```
POST /services/PlaceLookup.asmx HTTP/1.1
```

```
Host: codebump.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://skats.net/services/GetPlaceDetails"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
   instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:tns="http://skats.net/services/"
   xmlns:types="http://skats.net/services/encodedTypes"
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <tns:AuthenticationHeader>
      <SessionID xsi:type="xsd:string">string</SessionID>
    </tns:AuthenticationHeader>
  </soap:Header>
  <soap:Body soap:encodingStyle=
   "http://schemas.xmlsoap.org/soap/encoding/">
    <tns:GetPlaceDetails>
      <place xsi:type="xsd:string">string</place>
      <state xsi:type="xsd:string">string</state>
    </tns:GetPlaceDetails>
  </soap:Body>
</soap:Envelope>
```

4.2 Difficulties for manual invocation

All the students found that invoking web services is not an easy task even when it is done manually. The task is much more difficult than writing a traditional program for a few reasons.

4.2.1 Lack of Documentation

Web services are designed as self-explanatory components in order to increase their interoperability. However, the information conveyed in WSDL is not adequate for a programmer to invoke the service.

The most difficult part in web service invocation is to determine the parameters of the operations. In conventional programming languages, there are always explanations for operations and parameters in the operation. Sometimes there are even sample codes to illustrate the usages of the classes. In addition, with the class structure and the relationship with other classes, it is easy to derive a conceptual model of the problem at the hand.

For WSDL description, in general there are no comments in the document element to explain the parameters and operations. In addition, it is hard to find the web page containing use instructions that corresponds to the WSDL.

Web services usually are coarse grained and standalone, with the overall structure of the software hidden behind. Although most WSDLs are automatically generated from conventional programs, it seems that the comments in programs are not carried over to WSDL. For example, when we run java2wsdl in Apache Axis, the comments in the Java program are filtered out.

4.2.2 Versions of WSDLs

Web services are notorious for their fast evolution. For example, eBay service has a fundamental release every two weeks. The constant change makes even experienced programmers difficult to invoke the service. Some students have to contact the service providers to figure out the correct parameters to send out, because existing documents are for the older versions. With different versions of WSDLs scattered around on the web, web service invocation is like exploring a labyrinth. The situation is excerbated with the scarcity of documentation for web services and the connection between them.

Since conventional software has mature version control and upgrading system, web services need to have a proper management system as well.

5. Web service composition

In the experiment we found the difficulties of service composition can be classified into the following categories: schema mapping between data types of web services, large data problem, data quality and optimization, and volatility of web services.

5.1 Schema mapping

In web service composition literature, it is common to assume that we can use one service's output as another's input. In our experiment, the most common problem reported is the disparity of schemas between web services to be composed. Web services are developed by different organizations that use different conceptual models and vocabulary. Inevitably the resulting schemas in web services are different even if they are meant to be similar or the same.

Example 2: We have two services *getAirport* and *getAirportCoordinate* to be combined: *getAirport* is of the type

state→airport(code, city, country, name)

and *getAirportCoordinate* is of the type

airportCode→(latitude, longitude).

In this case, we need to map the output data

<airport> <code/>…<name/></airport>

from one service to the input data <*airportCode/*> of another service.

This kind of schema mapping is not easy to be automated [6]. There are substantial researches on XML Schema mapping, with the aim to identify the correspondences between the elements of schemas, so that the data can be integrated. Schema mapping is also actively studied in migrating legacy systems. In web service composition, every web service is like a legacy system: it is developed by a third party that does not have adequate documentation. Web service composition is similar to legacy system integration, where we need to build the mappings between the parameters of the services. In our experiment, it is almost never possible to directly use one service's output as another's input without any change of the data.

5.2 Large data and quality of data

Several web services in our experiment return huge number of data, which stalls the composition program, and makes the composite service inefficient and practically not possible to run.

Example 3: Searching airports by country will result very large data, which makes the composite web service practically impossible to run. Hence we have to select an alternative, i.e., obtaining the airports in a state instead of a country.

Another common problem is the low quality of the data returned. Quite often, we need to write some code to process the data before it can be passed to another service.

Example 4: Although the return of the Airport web service is an XML document, it is not well formed. As a result, it could not be validated using any XML schema. The problem with the XML document was that, the GMT Offset tag was missing in several airport elements. On top of that, it returns each airport information twice.

5.3 Optimization of composite service

Some web service composition seems perfect logically, and the data transferred is not big. Still, the composite service is extremely inefficient. The main reason is the involvement of remote invocation. While there are plenty of work on programming code optimization and database query optimization, there is little investigation on the optimization for composite web services.

Example 5: Given two services
book(Keywords?, Price, Currency),
and
exchange(FromCurrency?, ToCurrency?, FromAmount?, ToAmount).

The first returns the price and currency when given as input the keywords of the books. The second service accepts the amount the currencies to be converted, and gives as output the equivalent amount in another currency. It is easy to generate a composite service that gives the price in local currency:

localBook(Keywords?, LocalCurrency?, LocalPrice)
:- book(Keywords?, Price, Currency),
exchange(Currency?, LocalCurrency?, Price?, LocalPrice).

Straightforward implementation of this composite web service will need to invoke exchange services as many times as the number of returns of the book search. For each *Price* for a book, the exchange service is called to get the corresponding *LocalPrice*.

However, one call to the exchange service is enough. To optimize this composite service, we need to reformulate the composite service as below, so that the calls to web services can be minimized:

```
localBook(Keywords?, LocalCurrency?, LocalPrice)
  :- book(Keywords?, Price, Currency),
     exchange(Currency?, LocalCurrency?,   1, X), LocalPrice=X*Price.
```

5.4 Sporadic and inactive web service

Almost all web services, including commercial ones, are not constantly available during our experiment period that lasted two months. Some times they could be off line for a few days. For example, *Medicare* and *YahooTravel* services experienced two days blackout.

Although it is common for a web site to be off line for a while for maintenance or network disruption reasons, the only affected place is the web site itself. Volatile web services will ripple its unreliability throughout all the applications that use them.

To engineer a robust composite web service, it is paramount to have backup web services. Almost all the groups built backup service just to make sure that the composite web service would work properly on the demonstration day. For functions that are served by several services, such as text messenger, we keep all of them. For functions that are provided by only one vendor, such as airport information, students replicate the service on our own machine by downloading part of the data and deploy our own service, so that it can be used as a contingent plan.

Sporadic and inactive web services impose a serious problem for composite web services, as there are abundant inactive web services on the web. Before trying out the web services, there is no way to tell whether they can be used. Table 3 lists the ratio between active web services from two groups. Please notice that most web services are from service portal. If the WSDLs are collected from UDDI or google, the activeness ratio would be much lower.

Table 3: Ration of active web services

	WSDLs checked	Active web service	Active Web Service Ratio
Group 1	11	5	0.45
Group 2	14	9	0.64

6. Conclusions

This is the first empirical study that we are aware of on web service composition based on publicly available real web services. Since this study is based on real web services available, the semantic approach to web service annotation, discovery, and composition is not covered. There have been substantial empirical studies on the state of the art of web services [5] [22], but not the compositions. We programmed using more than 100 web services, and constructed 12 composite services. Some composite services contain several operations.

The study shows that web service composition is a creative activity, whose automation is a daunting task that requires the efforts from service provider as well as service consumers.

Service providers need to develop and maintain high quality web services in a disciplined and collaborative way. In particular, providers need to

1) *Apply software engineering principles*: Most web services are developed in an ad hoc manner, disregarding all the software engineering principles. Even tools developed for web service generation ignored the importance of documentation. For example, Apache Axis removed comments in Java classes when generating WSDLs from those classes.
2) *Develop services collaboratively*: Collaborative web services need to be developed collaboratively. For example, if an XML Schema is already developed or used in other web services, it should be reused, instead of being reinvented every time a similar schema is needed. This way, schema heterogeneity can be minimized. Just as writing conventional programs, developers should reuse existing components instead of re-develop similar classes and functions every time you need it. To achieve this goal, it is paramount to build and maintain a repository for web services and schemas to enhance the reuse.

Partially due to the inadequacy of WSDLs observed above, some students in our experiment preferred programmable web APIs to create mashups [13]. Similar to web service composition, when programmable web APIs are selected, it is relatively easy to compose them, as the process has little difference from the conventional programming. The difficulty is how to recommend appropriate services, and how to support end-users to create composite services with minimal programming effort [19].

Acknowledgements We would like to thank all the students who also participated in the experiment, and the anonymous reviewers for their helpful comments. This work is supported by NSERC.

References

1. Agarwal, V., Dasgupta, K., Karnik, N., Kumar, A., Kundu, A., Mittal, S., and Srivastava, B. A service creation environment based on end to end composition of Web services. *Proceedings of the 14th international Conference on World Wide Web* 2005. pp. 128-137.
2. Benatallah, B., Hacid, M., Leger, A., Rey, C., and Toumani, F. 2005. On automating Web services discovery. *The VLDB Journal* 14, 1 (Mar. 2005), pp. 84-96.
3. Tevfik Bultan, Jianwen Su, Xiang Fu: Analyzing Conversations of Web Services. IEEE Internet Computing 10(1): 18-25 (2006)
4. Dustdar, S., Schreiner, W.: A survey on web services composition. Int. J. Web and Grid Services 1 (2005). pp. 1-30
5. Fan, J. and Kambhampati, S. 2005. A snapshot of public web services. *SIGMOD Rec.* 34, 1 (Mar. 2005), 24-32.

6. Jianguo Lu, Ju Wang, Shengrui Wang, XML Schema Matching, IJSEKE, International Journal of Software Engineering and Knowledge Engineering, in Press.
7. Jianguo Lu, Yijun Yu, John Mylopoulos, A Lightweight Approach to Semantic Web Service Synthesis, ICDE Workshop on Challenges in Web Information Retrieval and Integration, Tokyo, 2005.
8. Matskin, M. and Rao, J. 2002. Value-Added Web Services Composition Using Automatic Program Synthesis. In *Revised Papers From the international Workshop on Web Services, E-Business, and the Semantic Web*, 2002. 213-224.
9. McIlraith, S.A., Son, T.C.: Adapting golog for composition of semantic web services. In: Proc. of the 8th Int. Conf. on Principles and Knowledge Representation and Reasoning (KR-02), Toulouse, France. 2002
10. Medjahed, B., Bouguettaya, A., Elmagarmid, A.K.: Composing web services on the semantic web. The VLDB Journal 12 (2003) 333-351.
11. Milanovic, N., Malek, M.: Current solutions for web service composition. Internet Computing, IEEE 8 (2004)
12. Mao, Z.M., Brewer, E.A., Katz, R.H.: Fault-tolerant, scalable, wide-area internet service composition. Technical Report UCB//CSD-01-1129, University of California, Berkeley, USA (2001)
13. ProgrammableWeb, http://www.programmableweb.com.
14. E. Rahm, P. A. Bernstein. A survey of approaches to automatic schema matching. VLDB J., 10(4):334-350, 2001.
15. Ponnekanti, S.R., Fox, A.: SWORD: A developer toolkit for web service composition. In: Proc. of the 11th Int. WWW Conf., 2002.
16. Rao, J., Su, X.: A survey of automated web service composition methods. In: Proc. of the 1st Int. Workshop on Semantic Web Services and Web Process Composition, SWSWPC2004, LNCS, San Diego, USA. (2004)
17. E. Sirin, B. Parsia, and J. Hendler, Composition-driven filtering and selection of semantic web services, AAAI Spring Symposium on Semantic Web Services, 2004.
18. UDDI, http://www.uddi.org/
19. Wong, J. and Hong, J. I. 2007. Making mashups with marmite: towards end-user programming for the web. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (San Jose, California, USA, April 28 - May 03, 2007). CHI '07. ACM Press, New York, NY, 1435-1444.
20. Wu, D., Parsia, B., Sirin, E., Hendler, J.A., Nau, D.S.: Automating DAML-S web services composition using SHOP2. In: Proc. of the 2nd Int. Semantic Web Conf.(ISWC2003), Sanibel Island, FL, USA. (2003)
21. Yijun Yu, Jianguo Lu, Juan Fernandez-Ramil, Phil Yuan, Comparing Web Services with Other Software Components, International Conference on Web Services, International Conference on Web Services(ICWS), 2007. pp. 388-397.
22. Zhang, L., Chao, T., Chang, H., and Chung, J. 2003. XML-Based Advanced UDDI Search Mechanism for B2B Integration. *Electronic Commerce Research* 3, 1-2 (Jan. 2003), 25-42.
23. Zhang, R., Arpinar, I.B., Aleman-Meza, B.: Automatic composition of semantic web services. In: Proc. of the 2003 Int. Conf. on Web Services, 2003.