# A Lightweight Approach to Semantic Web Service Synthesis

Jianguo Lu
*School of Computer Science*
*University of Windsor*
*jlu@cs.uwindsor.ca*

Yijun Yu, John Mylopoulos
*Department of Computer Science*
*University of Toronto*
*{yijun, jm}@cs.toronto.edu*

## Abstract

*Web service technologies are becoming a new paradigm for distributed computing. With increasing number of web services available on the internet, there is an urgent need for information brokers that can autonomously integrate web services on behalf of a user. To this end, we propose web service specification, which defines what the service is but not how the service is implemented, and service synthesizer, which can dynamically synthesize the implementation of the specification from existing web services over the web.*

## 1. Introduction

Web service technologies, embodied in the key W3C standards WSDL [24], UDDI [23] and SOAP [22], are becoming a new paradigm for distributed computing and programming that have been adopted by major software vendors. In this paradigm, software components described in XML will be published, requested, composed and even brokered over the Internet.

The essence of web service is its capability to be consumed by computer programs and integrated with other functionality of the system, ideally, in the same form of web services. Web service composition has attracted attention from both industry and the academic communities. BPEL4WS [10] is an XML-based programming language that can compose web services manually. Many researchers also start to investigate the automated service composition, such as [13] [26], to name a few.

We propose an approach to automated dynamic web service synthesis. It is lightweight in the sense that we can only handle a portion of the web services that are generated from databases and enterprise beans. We observe that many web services are database query wrappers, with mappings between XML Schema and database schema. As a matter of fact, there are numerous products that generate web services from database application and enterprise beans [15]. For this type of web services, there should be a declarative and automated way to integrate them. Those web services can be viewed as a database query, and hence the integration of web services can be viewed as database query rewriting [11].

This paper describes a web service synthesizer which can produce composite web services from web service specifications. End-users or programmers will provide the specification. The service synthesizer will take the specification as input, and search, customize, and integrate constituent web services automatically. The essence of this paper is to treat web services as first class citizens, hence they can be integrated without resorting to process flow languages or programming languages such as Java. In this way, end-users can customize existing web services, and compose new services without using a programming language.

An application scenario is as follows: Suppose that you want to construct a new web service that compares prices from *amazon.com* and *chapters.com*, on the premise that there are already implemented web services for *amazon*, *chapters*, and *currency converters* on the web. Also notice that you may not know the exact service name, and the category they belong to. What you need to do is to write a web service specification, which consists of mainly the signature of the service similar to what is provided by WSDL, and a database query to carry out that task. The web service synthesizer will take that specification as input, find relevant web services, and produce an implemented web service.

In section 2 we will describe the background knowledge of query rewriting that is used in our paper. Section 3 defines web services specification. Based on this definition, Section 4 introduced web service synthesis and Section 5

describes how the concrete implementation is generated. Section 6 draws the conclusion.

## 2. Background

### 2.1 Query rewriting

The *query rewriting problem [11]* has been extensively studied in the areas of query optimization and data integration. Informally speaking, the problem can be formulated as follows: Given a query and a set of view definitions, compose an answer to the query using answers to the views.

**Definition 1** (*Query containment and equivalence*) A query Q is contained in another query Q′, denoted as $Q \subseteq Q'$, if for any instance of the base relations, the set of tuples computed for Q is a subset of those computed for Q′. Two queries Q and Q′ are equivalent (denoted as $Q = Q'$) if $Q \subseteq Q'$ and $Q \supseteq Q'$.

**Definition 2** (Q*uery rewriting*) Given a query Q and a set of views V, a *rewriting* of Q using V is a query Q′ such that Q = Q′, and Q′ refers to one or more views in V. A rewriting is complete if Q' only refers to views.

We should note that rewritings do not always exist.

### 2.2 Datalog notation

It is easier and a common practice to discuss query rewriting in terms of the Datalog notation. Datalog is similar to Prolog, but does not allow functions in Horn clause expressions. In the following sections of this paper we will use this notation.

A conjunctive query has the form:

$$Q(X) :- P_1(X_1), P_2(X_2), \ldots, P_n(X_n), C_1(Y_1), C_2(Y_2), \ldots, C_m(Y_m).$$

where Q, $P_i$ (i=1,…, n), and $C_j$(j=1,…,m) are predicate names. X, $X_1$, …, $X_n$, $Y_1$, …, $Y_m$ are vectors of variables and constants. Q(X) is called the head of the query. $P_i$ (i=1,…, n) refer to the relations in the database, and are called base predicates. $C_j$ ( j=1, …, m) are arithmetic comparison predicates.

## 3. Web service specification

Currently, most of the web service definition mechanisms such as WSDL define the syntax of the web services, such as the operations of the service, the signature of the operation, and the binding information that describes how to invoke the operation. The semantics of the operations in web services are left unclear. The lack of explicit semantic definitions of web services makes it hard to search for the relevant services, and impossible for automated synthesis of new services.

On the other hand, there are research efforts to add semantics to agent-like web services. One example is OWL-S[20], which is based on web ontology languages and software agent description language [19]. These description languages focus more on software agent concepts rather than WSDL, hence are more complex and farther away from industry applications.

We observe that a large portion of web services are based on database applications. Most of the database vendors (such as DB2) and EJB vendors are providing web service construction tools to expose SQL statements as web services[15]. Moreover, any web service can be consumed inside a database query by the DB2 web service consumer. Based on these observations, we propose to describe the semantics of web service using queries.

More formally, a web service specification is defined as follows:

**Definition 3** (*Web service specification*) A web service specification is S(Sig, Q), where Sig is the signature of a web service and Q is the corresponding query of the web service.

The signature of web service specification consists of the input and output types, which are defined using XML Schemas much in the same way as WSDL. Note that in this definition we simplify the model and assume web services can have only one operation. Also, we abstract away the binding information which is essential for web service implementation.

Web service specification defines what the service is, but not how the service is implemented. Given a specification, it can be implemented in different ways, such as by a database engine or an EJB server.

With the understanding that a query can be represented by Datalog, our definition of web service specification share similarities with program specifications. Compared with traditional program specification techniques, such as pre/post conditions and algebraic specifications, our approach has direct correspondence to the underlying implementation. For example, IBM DB2 provides WORF [15] to specify the web service and its related query. From a DADX specification, a WSDL file can be automatically generated.

Web service specification serves two groups of people. One is the service providers who want other people to use the web services. In this case, the specification has a corresponding implementation. By providing specification, web services can be more effectively located and reused. The other group of people is service consumers and composer. When they create a new web service, what they need to write is a web service specification. In analogous to

traditional program synthesis, the implementation of the specification will be automatically synthesized.

It has always been a painful task to synthesize a program from its specification. Part of the difficulty comes from the lack of reusable software components. With the proliferation of web services, the synthesis can become easier as we show in this paper.

With this definition of web service specification, we can discuss web service synthesis in the following sections.

## 4. Web service synthesis

Given a web service specification, we need to produce the corresponding implementation for the specification. We call the process to generate a web service implementation from its specification the *web service synthesis*, which is carried out by *service synthesizer* in a dynamic way.

More specifically, the input of the synthesizer is a web service specification S to be implemented, and a set of web service specifications $\{S_1, S_2, ..., S_n\}$ which are already implemented. The output of the synthesizer is an implementation of S.

Intuitively, a web service specification S is implemented if its query can be executed directly by a data source, or, can be rewritten into several web services which are implemented. There are two kinds of implementations of a service. One is abstract implementation. S' is an abstract implementation of S if Q in S can be rewritten using the queries in service $S_1$, ...., or $S_n$. The concrete implementation comprises the scaffolding code that carries out the abstract implementation. An example of concrete implementation is illustrated later in figure 1.

Correspondingly, the synthesis is carried out in two steps. The first step is to synthesize an abstract implementation. Based on the abstract implementation, we can derive different concrete implementations.

**Definition 4** (*Abstract implementation*) Given a web service specification S(Sig, Q), and two web services S1(Sig1, Q1), S2(Sig2, Q2), an abstract implementation of S(Sig, Q) using S1 and S2 is S(Sig, Q') where Q' is a complete rewriting of Q using Q1 and Q2.

Once we obtained an abstract implementation of a web service, we can derive the concrete implementation in different programming languages.

We assume that there will be a global database schema, for all the existing web services that are based on databases. The creation of such a global schema is not covered in this paper. Based on this global schema, and the mapping between the database schema and XML schema, we can write web service specifications, i.e., the signature of the service and the query to be carried out. Taking the specification as input, the service synthesizer will search from existing services over the web for the services that can carry out the task, based on the signature and the semantic description. In the case of matching of the semantic part of the web service specification, the matching is reduced to the query containment testing, since the semantic annotations are represented by Datalog. If no such service can be found, the synthesizer will decompose the specification into sub specifications in the hope that the sub-services are ready on the web. Once all the sub-services are found, the resulting rewriting will be planned and according to the executable planning, the composition code is generated to reproduce the functionality of the original query.

The drawback of our approach is that not every query can have a complete rewriting, hence not every web service specification is implementable. However, with the proliferation of web services, more and more web service specifications will become implementable.

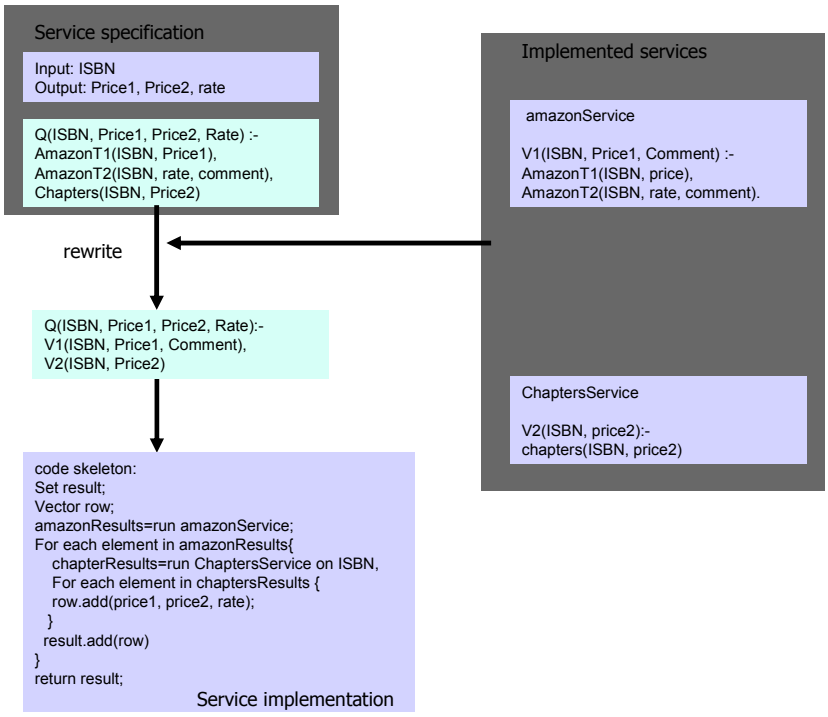


**Figure 1: Example of web service synthesis**

Figure 1 illustrates the service synthesizer using a concrete example. Suppose there are *Amazon* service and *Chapters* service, both are implemented and annotated with the underlying queries. We can write a new service specification called *metaBookService*. The input of the *metaBookService* specification is an ISBN, and the output of the specification consists of the price and book rating from *amazonService*, and the price from *chaptersService*. In addition to the signature description of the service, there is also a logic specification for the operation, i.e., the join operation between the three database tables in two different sites.

Taking the specification as input, the service synthesizer will first search over the web to find relevant services. The search process will be based on several criteria such as the description of the web services, the similarities based on the signature, and the relevance between the queries. An earlier version of the searching mechanism is described in [19]. In our case, we are supposing the search result consists of *amazonService* and *chaptersService*.

Once those two services are ready, the next step is to try to find a complete rewriting of the query in *metaBookService* using the queries in *amazonService* and *chaptersService*. The result of the rewriting constitutes an abstract implementation of the *metaBookService*.

With the abstract implementation in hand, the service synthesizer can create a plan to execute the two existing services and scaffolding code to integrate the results from different services.

One of the key components of service synthesizer is service matching. When the number of services is large, it is not obvious how to locate two relevant services whose queries can be used in rewriting. With the Datalog as the semantics of the web service, the matching problem is reduced to query containment problem which is well studied.

```
create function bn_price (isbn varchar(10))returns varchar(10)
language sql reads sql data external
action not deterministic
return with
soap_input (in) as values varchar(xml2clob(
xmlelement(
  name "rns:getprice",
  xmlattributes(
   'urn:xmethods-bnpricecheck' as
               "xmlns:rns"),
  xmlelement(name "isbn", isbn))))),

soap_output(out) as (values db2xml.soaphttpc
('http://services.xmethods.net:80/soap/servlet/rpcrouter', '',
(select in from soap_input)))
 select cast(a.returnedchar as
             varchar(10)) as price
 from table(db2xml.extractchars((
     select cast(out as db2xml.xmlclob)
     from soap_output), '/*/return')) as a;
```

**List 1: DB2 UDF for Barnes Noble**

## 5. The concrete implementation of web service

Once a complete rewriting is obtained for the query in web service specification, we need to generate a plan for the query so that program code can be generated to actually execute the query. A query plan is a sequence of accesses to the web services interspersed with local processing operations. Given a query Q of the form:

$$Q(X) :- V_1(X_1), \ldots, V_n(X_n).$$

A plan to answer it consists of a set of conjunctive plans. Conjunctive plans are like conjunctive queries except that each subgoal has input and output specification associated with it. For example, a plan for the above query could be

$$\begin{aligned} Q(X) :- \; & V_1(X_1)(In_1, Out_1), \\ & V_2(X_2)(In_2, Out_2), \ldots, \\ & V_n(X_n)(In_n, Out_n). \end{aligned}$$

A plan is executable if the input of the i-th predicate appears in the output of the preceding predicates, i.e., $In_i \subseteq Out_1 \cup \ldots \cup Out_{i-1}$

Once we generated the execution plan, we can replace the views with service invocations, and provide the input parameters using input/output definitions.

To illustrate, we sketch a web service implementation using IBM DB2 [40]. In DB2, RDB (Relational Data Base) data and SOAP data are exchanged in two ways: WORF or DB2 web services consumer (DB2WS) [15]. DADX (Data Access Definition) is an XML extender that wraps up RDB tables or SQL queries into web services and maps XML data into RDB tables; DB2WS is a set of stored procedures that supports User-Defined Functions (UDF) that consume a web service as if it is a local function.

For our purpose, we adopt DB2WS to invoke web services based on their web service specifications only. After composing the web services, we can use the DADX to publish the data as a composed web service.

For example, to consume the Barnes & Noble web service, we declare a DB2 UDF based on the web service published at the *xmethods* site as in List 1.

The user-defined function in DB2 sends a request SOAP message to the web service using the input parameter provided by the function, e.g. ISBN, and receives a response SOAP message from the web services, which is parsed into a temporary table using XPath expressions, e.g. "/*/return". The parsed result is further selected in SQL as an output field, e.g. "price". Along the round-trip, at the client-side, DB2XML stored procedures are used as marshalling mechanism for the SOAP messages. To the caller, such a user-defined function appears the same as a function querying the price from a local table, as the service scaffolding code are buried in its implementation.

Both input and output of the function *BN_PRICE* are in string format. In this function, the input *ISBN* string is used to compose SOAP request with an envelope in XML format. The SOAP response from the Barnes & Noble web service is parsed into a *PRICE* string by an extractor function based on the XPath "*/*/return"*. Thus, the Barnes & Noble web service can be invoked as if an internal user-defined function. The following is a use scenario of this UDF.

```
drop table books;
create table books(
          isbn varchar(10) not null);
insert into books values ('0439139597');
insert into books values ('0792386663');
select b.isbn, bn_price(b.isbn) as price
from books b;
```

Similarly we can wrap up the Amazon web service as the following UDF. This web services take the ISBN as the input and generates a tuple (bookname, ourprice, rating) as the output.

```
drop function amazon_price;
create function amazon_price(
         isbn varchar(20))
returns table (
    bookname varchar(108),
    ourprice varchar(10),
    rating varchar(180)
   )
  language sql reads sql data external action not
deterministic
return with
soap_input (in) as (values varchar(xml2clob(
xmlelement(name "typens:AsinSearchRequest",
xmlattributes('http://soap.amazon.com'
           as "xmlns:typens"),
xmlelement(name "AsinSearchRequest",
  xmlattributes('typens:AsinRequest' as
                            "xsi:type"),
xmlelement(name "asin", isbn),
xmlelement(name "page", '1'),
xmlelement(name "mode", 'books'),
xmlelement(name "tag", 'webservices-20'),
xmlelement(name "type", 'heavy'),
xmlelement(name "devtag", 'D2Y0B3SQUFMPKI'))))))),

soap_output(out) as (values db2xml.soaphttpc
('http://soap.amazon.com/onca/soap2',
'http://soap.amazon.com',
(select in from soap_input)))
  select *
  from table(tableextract((
    select cast(out as db2xml.xmlclob)
    from soap_output),
   '/*/return/Details/Details/',
   'Productname',
   'OurPrice',
   'Rating')) as x;
```

In the above implementation, the parsing of a more complex response SOAP message body is done through a generic UDF function "tableextract" where any parameter names can be used to extract selectively several tags into a table. To make sure the "tableextract" function is generic, intermediate single column views were created using a DB2XML stored procedure, and they are selectively joined according to the arguments of "tableextract". Here another UDF "tableextract" is required to extract the SOAP response into a table.

```
drop function tableextract;
create function tableextract(x db2xml.xmlclob,
    root varchar(50),
    p1 varchar(50),
    p2 varchar(50),
    p3 varchar(50))
  returns table(bookname varchar(250), ourprice
      varchar(10),
      rating varchar(180))
language sql reads sql data no external action
not deterministic
return  (
 select f1, f2, f3
 from
  (select row_number() over () as no,
  cast(t1.returnedchar as varchar(250)) as f1
  from table(db2xml.extractchars(x,
    concat(root,p1))) as t1) as t1,
  (select row_number() over () as no,
    cast(t2.returnedchar as varchar(10))
                                      as f2
   from table(db2xml.extractchars(x,
             concat(root,p2))) as t2) as t2,
   (select row_number() over () as no,
           ca st(t3.returnedchar as
                      varchar(18)) as f3
    from table(db2xml.extractchars(x,
              concat(root,p3))) as t3) as t3
where t1.no = t2.no and t1.no = t3.no);
```

These UDFs (Amazon and BN) in DB2 can be regarded as the concrete implementation of their web service specifications. They can be synthesized into a composite web service simplify by implementing its abstract web service specification, that is, the complete query rewriting of the two constitute web service specifications in Figure 1. The following SQL query is an example of such implementations.

```
select b.isbn, a.ourprice,
       bn.price, a.ratings
from books b,
  table(amazon_price(b.isbn)) as a,
  table(bn_price(b.isbn)) as bn;
```

## 6. Conclusions

Thanks to the widespread use of web based information systems, and the introduction of industrial standards such as WSDL, UDDI, and SOAP, there is a growing demand for integrating web services dynamically.

Web service composition can be classified into two categories, i.e., manual and automated. There are numerous industry efforts in manual composition, leading by web service composition language BPEL[10]. Automated composition replies on the semantic specification of web services. There are substantial researches on semantic web services [21].

With the understanding that web service is a view definition of the underlying database, new web services can be defined using a query over global views and implemented using query rewriting.

Unlike many information integration systems that have their roots in heterogeneous database systems, we adopt a

different approach where each information source is described as a function, rather than a database schema. Moreover, such functions take XML documents as input and produce XML documents as output.

The main contributions of this work are as follows. Firstly, we propose a web service specification which can handle a large portion of existing web services. Secondly, we define web service synthesizer which can dynamically generate the implementation of a service specification. We have implemented the core components of the synthesizer and matcher, i.e., the query matching and query rewriting, and also experimented the mapping between database queries and web services described in WSDL.

The obvious limitation of our approach is that our specification language is Datalog, which is not expressive enough to describe the semantics of all kinds of web services. One example is that it has difficulty in describing tree structured XML data.

Another constraint of our approach is the assumption of the existence of global schemas that both the service provider and the service specifier shall know and share. If we view web services as information sources, and the new web service specification as a query in the information mediator, our approach can regarded as a Local-As-View approach in data integration[11].

Another practical consideration of our approach is whether the synthesizer can always find the implementation for a specification. The viability of approach depends on the availability of a large amount of web services annotated with Datalog semantics, so that whenever people type in a Datalog specification for a web service, our synthesizer would be able to find the relevant services, and compose them accordingly. To achieve this goal, we need to develop efficient web service search engines, in the vein of software agent searching as in [19]. Since one of the major components in web service specification is the types of the operations, which are defined in terms XML Schemas, we have developed a mechanism to match XML Schemas[13]. Based on this, we are developing a web service matching system.

## Acknowledgement

The authors would like to thank the supports from NSERC (Natural Sciences and Engineering Research Council of Canada), IBM, and CITO (Communications and Information Technology Ontario).

## References

[1] W. van der Aalst, Don't go with the flow: Web services composition standards exposed, *IEEE Intelligent Systems*, Jan/Feb 2003.

[2] M. Aiello et al. A Request Language for Web-Services Based on Planning and Constraint Satisfaction. In *VLDB Workshop on Technologies for E-Services* (TES02), 2002.

[3] Boualem Benatallah, Mohand-Said Hacid, Christophe Rey, Farouk Toumani: Request Rewriting-Based Web Service Discovery. *International Semantic Web Conference* 2003: 242-257

[4] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, et al, Simple Object Access Protocol (SOAP) 1.1, *W3C Note,* May 2000.

[5] Paul A. Buhler José M.Vidal, Toward the Synthesis of Web Services and Agent Behaviors, In Proceedings of the *First International Workshop on Challenges in Open Agent Systems*, pages 25-29, 2002.

[6] D. Florescu and D. Kossmann. XL: An XML Programming Language for Web Service Specification and Composition. *Proceedings of the eleventh international conference on World Wide Web*, 2002.

[7] W-J Van Heuvel, J. Yang, and M.P. Papazoglou. Service Representation, Discovery, and Composition for E-Marketplaces, Proc. *of International Conference on Cooperative Information Systems* (COOPIS 01), Sep, 2001.

[8] Snehal Thakkar, Jose-Luis Ambite, and Craig A. Knoblock. A view integration approach to dynamic composition of web services, In *Proceedings of 2003 ICAPS Workshop on Planning for Web Services*, Trento, Italy, 2003.

[9] H. Hosoya and B. C. Pierce, XDuce: A Typed XML Processing Language, *Int'l Workshop on the Web and Databases (WebDB)*, Dallas TX, 2000.

[10] IBM, BPEL4WS, Business Process Execution Language for Web Services, Version 1.0, July 2002

[11] Alon Levy, Answering queries using views: a survey, *VLDB Journal* 2001.

[12] Jianguo Lu, John Mylopoulos, XIB: eXtensible Information Broker, *International Journal on Artificial Intelligence Tools, Vol. 11, No. 1*, March 2002. p.95-115.

[13] Jianguo Lu, Shengrui Wang, Ju Wang, An Experiment on the Matching and Reuse of XML Schemas, *International Conference on Web Engineer*ing, Sydney 2005

[14] S. McIlraith and TC Son. Adapting Golog for Composition of Semantic Web Services, *KR*'2002.

[15] Malaika, S. et al. DB2 and Web Services. *IBM System Journal*, 41(4), pp. 666-685. 2002.

[16] Tova Milo and Dan Suciu and Victor Vianu, Typechecking for XML Transformers, *19th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems,* ACM, 2000. p.11-22.

[17] Paulo F. Pires and Mario Benevides and Marta Mattoso, Building Reliable Web Services Compositions, Net.Object Days - WS-RSD'02,551-562, 2002.

[18] Evren Sirin, Bijan Parsia, Dan Wu, James A. Hendler, Dana S. Nau: HTN planning for Web Service composition using SHOP2. *J. Web Sem*. 1(4): 377-396 (2004).

[19] Katia Sycara, Jianguo Lu, M. Klusch, Advertisement and Matchmaking among Information Agents, *Technical Report* CMU-RI-TR-98-22, CMU.

[20] K. Sycara, M. Klusch, S. Widoff, Jianguo Lu, Dynamic Service Matchmaking among Agents in Open Information Environments, *Journal of ACM SIGMOD Record,* Special Issue on Semantic Interoperability in Global Information Systems, A. Ouksel, A. Sheth (Eds.), 28(1):47-53, 1999.

[21] OWL-S Coalition, OWL-S specification, 2004.

[22] W3C, SOAP, www.w3c.org.

[23] W3C, UDDI.org UDDI Technical White paper, *http : //www.uddi.org/pubs/lru UDDI Technical Paper.pdf,* 2001.

[24] W3C, Web Service Definition Language. http://www.w3.org/TR/wsdl.

[25] WordNet -- A Lexical Database for English. http://www.cogsci.princeton.edu/wn/.

[26] L. Zeng, Dynamic Web Services Composition, *PhD thesis*, *Univ. of New South Wales*, 2003.