Dynamic Web Service Composition: A New Approach in Building Reliable Web Service

Pat. P. W. Chan and Michael R. Lyu
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Hong Kong, China
pwchan, lyu@cse.cuhk.edu.hk

Abstract

The use of services, especially Web services, became a common practice. In Web services, standard communication protocols and simple broker-request architectures are needed to facilitate exchange of services, and this standardization simplifies interoperability. In the coming few years, services are expected to dominate software industry. There are increasing amount of Web services being made available in the Internet, and an efficient Web services composition algorithm would help to integrate different algorithm together to provide a variety of services. In this paper, we provide a dynamic Web service composition algorithm with verification of Petri-Nets. Each Web service is described by Web Service Definition Language (WSDL) and their interactions with other services are described by Web Service Choreography Interface (WSCI). Our algorithm compose the Web services with the information provided by these two descriptions. After the composition, we verify the Web service to be deadlock free with modeling the Web service as a Petri-Net. We conduct a series of experiments to evaluate the correctness and performance of the composed Web service.

1. Introduction

Service-oriented Architectures (SOA) [9] are based on a simple model of roles. Every service may assume one or more roles such as being a service provider, a broker or a user (requestor).

The use of services, especially Web services, became a common practice. In Web services, standard communication protocols and simple broker-request architectures are needed to facilitate exchange (trade) of services, and this standardization simplifies interoperability. In the near future, services are expected to dominate software industry. There are an increasing number of Web services available in the Internet. Web services can be a component of a system and different Web services would provide different services. To fit different requirements from different clients, different Web service components can be combined together to provide the services. To achieve this, an efficient Web service composition algorithm is important.

Several Web service composition approaches have been proposed for Web services in the literature [4, 8]. Most of the existing algorithms are aggregating the Web service in a static approach, that is, making the composition after all the Web services are available. However, as the number of Web services is increasing, this would make the approach inflexible and hard to scale. In this paper, we propose a dynamic Web service composition algorithm which attacks these issues in a unified approach.

The rest of the paper is organized as follows. Related work of Web service composition is presented in Section 2. In Section 3, our composition algorithm is described, in which an example is employed to illustrate our approach. Experiments and the results are presented in Section 4. Finally, conclusions are made in Section 5.

2. Related Work

There are a number of techniques to enable the composition of Web services. The Web Service Choreography Interface (WSCI) [2] is an XML-based interface description language that describes the flow of messages exchanged by a Web Service participating in choreographed interactions with other services. WSCI describes the dynamic interface of the Web Service involved in a given message exchange by means of reusing the operations defined for a static interface. WSCI works in conjunction with the Web Service Description Language (WSDL), the basis for the W3C Web Services Description Working Group. It can also work with other service definition languages that exhibit the same

1550-445X/08 $25.00 © 2008 IEEE
DOI 10.1109/AINA.2008.133

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on December 30,2020 at 06:30:58 UTC from IEEE Xplore. Restrictions apply.

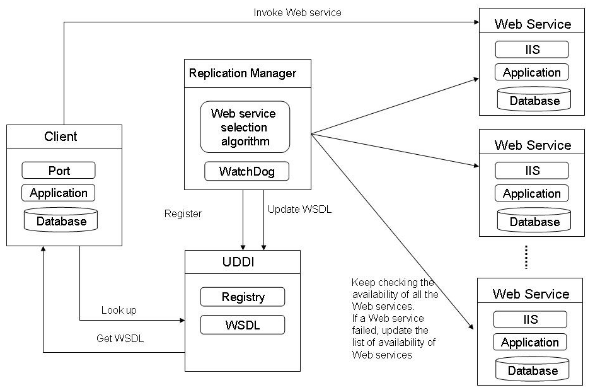

Figure 1. Proposed architecture for dependable Web services.

characteristics as WSDL.

Business Process Execution Language (BPEL) [1] for Web services is an XML-based language designed to enable task-sharing for a distributed computing or grid computing environment even across multiple organizations, using a combination of Web services. Written by developers from BEA Systems, IBM, and Microsoft, BPEL combines and replaces IBM's Web services Flow Language (WSFL) [10] and Microsoft's XLANG specification.

Moreover, a number of Web service composition schemes are proposed. SWORD is one of the proposed solution. It is a set of tools for the composition of a class of Web services including "information-providing" services. In SWORD, a service is represented by a rule to express that with given certain inputs, the service is capable of producing particular outputs. A rule-based expert system is then employed to automatically determine whether a desired composite service can be realized with the existing services.

When large-scale Web services are available, Chen et al. propose a structure to handle the composition. Then the mutual search operations among Web Service operations, inputs and outputs are studied, and a novel data structure called Double Parameter Inverted File (DouParaInvertedFile) is proposed to implement these operations. An algorithm to build DouParaInvertedFile is provided as well.

Apart from the self-contained algorithm, some algorithms are based on the current existing standards. In [8], a BPEL-based Web service composition using high-level Petri-Nets (HPN) approach is proposed. By analyzing the structure of Web service composition based on BPEL, the corresponding HPN is constructed. The dynamism and occurrence are presented in HPN with guard expression with colored token. After translation, the equivalent HPN of the Web service composition based on BPEL can be verified on existing mature tools.

Although a number of approaches have been proposed to aggregate the Web service, there is a need for a dynamic approach to compose the increasing number of Web services to provide new services. In this paper, we aim at proposing an innovative dynamic Web service composition algorithm.

3 Web Service Composition

Diversity is one of the key elements in the proposed paradigm. In the emergence of service-oriented computing, different versions of Web services or even different versions of their components are abundantly available in the Internet. The combination of different versions of the Web service or their components is thus becoming critical for enabling different versions in a server application using the N-version approach [11]. In our pervious work [4, 5, 6, 7], we propose a Web service paradigm for improving the reliability of the system. We describe the methods of dependability enhancement by redundancy in space and redundancy in time. The architecture of the proposed system is shown in Figure 1. In the system, the Web servers work concurrently and a Round-robin algorithm [14] or N-version programming is applied for scheduling the work among the Web services. The Web services run on different machines. When there is a Web service fault, the other Web servers can immediately provide the required service. The replication mechanism shortens the recovery time and increases the availability of the system. The Web services are coordinated by a Replication Manager, which schedules the workload of the Web services and keeps updating the availability of each Web service. To evaluate the reliability of Web services, we perform a series of experiments employing several replication schemes and compare them with a non-redundant single service. In the experiment, we find that N-version programming is one of the efficient method to improve the reliability of the system. Thus, in this section, we propose an approach for composing Web services with an N-version Programming Web for improving the reliability of the overall system.

3.1 Web Service Description

The description of a Web service is statically provided by WSDL, including Web service functional prototypes. However, its static nature limits the flexibility for composing Web services. Different Web services provide their services at different times, and so a dynamic composition approach is necessary for composing different versions of Web services available in the Internet.

In Web services, the communication mainly depends on the messages exchanged between different Web servers.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on December 30,2020 at 06:30:58 UTC from IEEE Xplore. Restrictions apply.

The Web Service Choreography Interface (WSCI) [2] is an XML-based language for the description of the observable behavior of a Web service in the context of a collaborative business process or work-flow.

3.2 The Proposed Composition Method

Our proposed service composition method is based on two standard Web service languages: WSDL and WSCI. WSDL describes the entry points for each available service, and WSCI describes the interactions among WSDL operations. WSCI complements the static interface details provided by a WSDL file describing the way operations are choreographed and its properties. This is achieved with the dynamic interface provided by WSCI through which the inter-relationship between different operations in the context of a particular operational scenario.

The flow of the composition procedure is as follows: First, get the WSDL of the Web service components from UDDI. Then, through the messages between the Web services, obtain the WSCI of the components. Afterwards, examine the input and output of the components through WSDL and determine the interactions between different components to provide the service through WSCI. Finally, perform the composition of the Web service with the information obtained in the composition procedure. The detailed composition algorithm is shown in Algorithm 1.

In Algorithm 1, we aim to build the tree for the Web service composition. We use a bottom-up approach to perform the composition, that is, we build the composition tree from output to input.

When we get the required output, search the Web services in the WSDL. In the operation tag of the WSDL, the output information is stated. When the desired output is found, that Web service component (CP_n) is inserted as the root of the tree. Then, if the input of that operation matches the required input, the searching is finished and the input is inserted as a child of the CP_n. Otherwise, we will search the action in WSCI in finding matches to the operation in CP_n. After the action is completed, we can determine the previous action. Then, we can find the operation prototypes in the WSDL. If the input of this operation matches the required input, then the composition is finished. Otherwise, we will iterate until the root of the WSCI is reached.

If the desired input is still not found, we will search for the operations in other WSDL whose output is equal to the input of CP_n. If the next Web service component found is CP_m, then CP_m is inserted as the child of CP_n. We perform the searching iteratively and continue to build the tree until all the inputs match the required input.

Algorithm 1 Algorithm for Web service composition

```
Require: $I[n]$: required input, $O[n]$: required output
 1: $CP_n$: the $n^{th}$ Web services component
 2: for all $O[i]$ do
 3:   Search the WSDL of the Web services, and find the
      $CP_n$ 's operation output = $O[i]$. Then, insert $CP_n$
      into the tree.
 4:   if the input of the operation = $I[j]$ then
 5:     Insert the input to the tree as the child of $CP_n$.
 6:   else
 7:     Search the WSCI of $CP_n$, WSCI.process.action =
        operation.
 8:     Find the previous action needing to be invoked.
 9:     Search the operation in WSDL equal to the action.
10:     if input of the operation = $I[i]$ then
11:       Insert input to the tree as the child of $CP_n$
12:     else
13:       go to step $(8)$
14:     end if
15:   end if
16:   until reaching the root of WSCI and not finding the
      correct input, search other WSDL with output = $I[j]$,
      insert $CP_m$ as the child of $CP_n$ and go to step (7) to
      do the searching in WSCI of $CP_m$.
17: end for
```

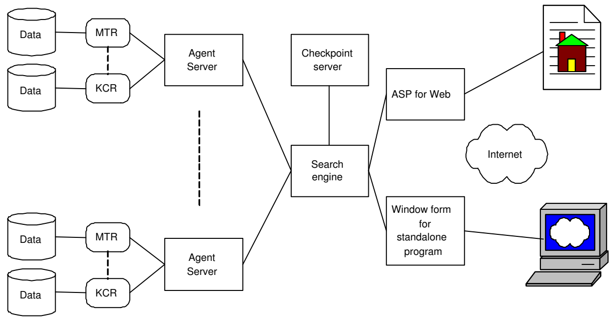

Figure 2. Best Route Finding system architecture.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on December 30,2020 at 06:30:58 UTC from IEEE Xplore. Restrictions apply.

3.3 Case Study

To illustrate the above procedure, we present the Web services composition with the Best Route Finding system (BRF) [3] whose architecture is shown in Figure 2. This system suggests the best route for a journey within Hong Kong by public transport, based on input consisting of the starting point and the destination. BRF consists of different components, including a search engine, agent servers, and the public transport companies. We acquired several versions of BRF, which are implemented by different teams using different components. Also, the Web service components may differ from versions to versions; thus, in this experiment, we try to compose the Web services from different versions with the WSDL and WSCI provided therein.

Also, the following shows part of the WSDL and WSCI specification of the search engine. The WSDL identifies the input and output parameters of the services provided by the search engine.

```
<?xml version="1.0" encoding="UTF-8"?>
 ...
<portType name=BRF">
  <operation name=shortestpath">
    <input message=
           "tns:startpointDestination"/>
    <output message="tns:pathArray"/>
  </operation>

  <operation name=addCheckpoint">
    <input message="tns:pathArray"/>
    <output message=
           "tns:addAcknowledgement"/>
  </operation>
  ...
  </operation>
</portType> </definitions>
```

The following shows part of the WSCI specification of the *search engine*.

```
<correlation name=pathCorrelation
property=tns:pathID></correlation>

<interface name=busAgent>
  <process instantiation="message">
    <sequence>
        <action name="ReceiveStartpointDest
           role="tns:busAgent
           operation="tns:BRF/shortestpath">
           <correlate correlation=
              tns: pathCorrelation/>
           <call process=tns:SearchPath/>
         </action>
           ...
      </sequence>
    </process>
...
```

Based on Algorithm 1, a composition tree is built , giving the result as shown in Figure 3.

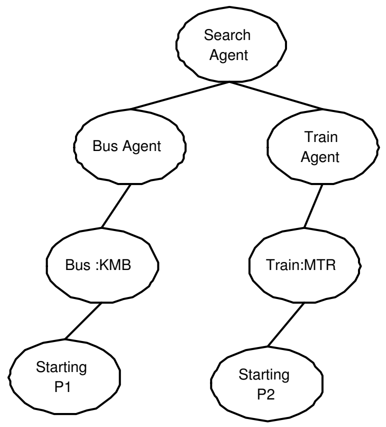

Figure 3. Composition tree of BRF.

3.4 Verification with Petri-Net

To verify the correctness of the composed Web service, Petri-Net [13] is employed. We first construct a Petri-Net for the Web service with the information provided in BPEL.

3.4.1 BPEL and Building Block of Petri-Net

After a Web service is composed with the proposed Algorithm 1, a BPEL is constructed. BPEL describes the composition properties of the Web service, such as communication and specific behaviors.

In the verification process, we employ Petri-Nets to build the model of the Web service to prevent deadlock and construct dynamic relations. Different building blocks of Petri-Nets are defined according to the activities in BPEL schema, including inner-service, intra-service, inter-activity, and intra-activity. With the defined blocks, we map the operations or activities specified in BPEL to the Petri-Net building blocks. Then, a Petri-Net for a specified Web service is generated.

A Web service operation is composed of basic activities (including Receive, Reply, Assign, Invoke, Empty, Terminate, and Wait) and structures activities (including While, Switch, Sequence, Link and Flow. The sample basic activities translation are shown in Figure 4. With the activities in BPEL, Web services are described procedurally. In a Petri-Net, a *place* connected to a *transition* intuitively expresses

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on December 30,2020 at 06:30:58 UTC from IEEE Xplore. Restrictions apply.

the *states* before and after executing the corresponding *action*. *Firing* a transition means that the corresponding action is executed. Moreover, Web service invocation is expressed by entering a *token* in a place which denotes the *starting point* of the operation.

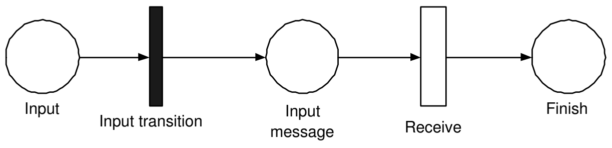

Figure 4. Basic Petri-Net building block – Receive.

Figures 4 illustrate the basic Web service operations composited with Petri-Net building blocks. A *building block* is presented by a place with a token whose type is specified by the block type. An arc is used to link the transition with another arc corresponding to the input or output message consisting of those blocks.

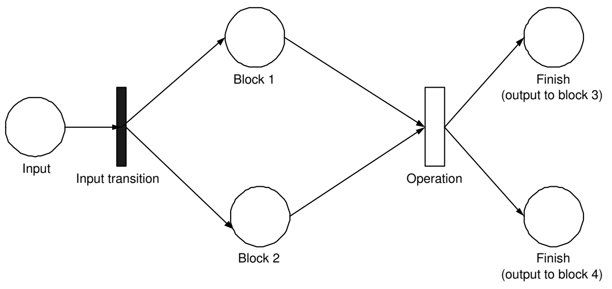

Figure 5. Composited Petri-Net building block graph.

With the Petri-Net building blocks and the BPEL of the BRF, Petri-Nets of different versions of BRF can be generated. One of the composited BRF is shown in Figure 6. With the constructed Petri-Net, we perform the operation to check the correctness and verify that the Web service is deadlock-free.

4 Experiment

In this section, we preform experiments to evaluate the properties, correctness and performance of the proposed Web service composition algorithm. We generate different versions of BRF with the Web service composition algorithm and evaluate with program metrics. Furthermore, we perform an acceptance test on the composed version to evaluate the correctness of the algorithm.

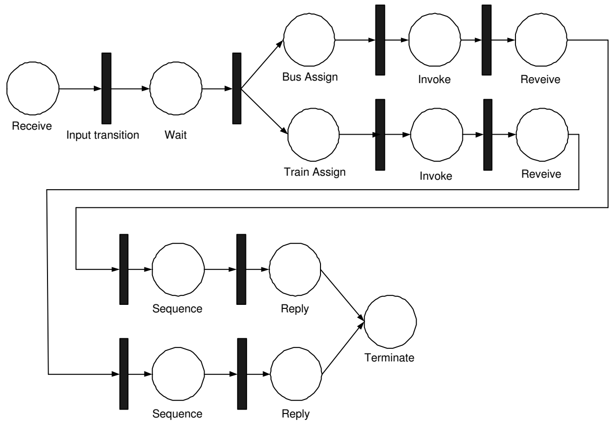

Figure 6. The Petri-Net of a BRF.

4.1 Different versions of Best Route Finding

We obtained several versions of BRF, which are implemented by different teams using different components. Also, the Web service components may differ between parties; thus, in this experiment, we try to compose the Web services from different versions with the WSDL and WSCI provided to create new versions.

According to the Web service composition algorithm described in Section 3, different versions of BRF are composed. The program metrics for 15 versions of BRF are shown in Table 1, where the first 11 versions are implemented by different teams and the rest are composed by the proposed algorithm. In the experiment, we recorded the following information of each versions: number of lines in the program, number of lines without comments, number of functions, complexity [12], composition time (the time for composing the version), and deadlock-free checking result.

4.2 Acceptance test

To test the correctness of the composed Web service, an acceptance test is prepared. Once the composition is finished, 100 test cases are run on the system. For the BRF, the acceptance test is designed as follows.

There are 100 tests cases. In each test, we provide a start point and a destination. The system will give out the best route with the transit points, total price and time. For each test case, we have the solution with which we compare the output from the system.

4.3 Discussion

After composition, a BPEL for that version of Web service would be created and the corresponding Petri-Net is

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on December 30,2020 at 06:30:58 UTC from IEEE Xplore. Restrictions apply.

Table 1. Program metrics of the 15 versions

ID	Lines	Line without comment	Number of function	Complexity	time for composition (s)	Deadlock free	Acceptance test
01	3452	3052	59	64	-	yes	pass
02	2372	1982	47	87	-	yes	pass
03	2582	2033	26	45	-	yes	pass
04	3223	3029	78	124	-	yes	pass
05	2358	2017	34	89	-	yes	pass
06	4478	3978	56	107	-	yes	pass
07	1452	1320	38	46	-	yes	pass
08	5874	5275	80	124	-	yes	pass
09	3581	3214	45	74	-	yes	pass
10	4578	4187	47	113	-	yes	pass
11	2364	2015	36	76	-	yes	pass
12	2987	2336	65	147	1.48	yes	pass
13	4512	3948	75	155	1.74	yes	pass
14	3698	3247	60	192	1.58	yes	pass
15	4185	3856	34	88	1.62	yes	pass

constructed for deadlock free-checking. The result is also shown in Table 1. With our proposed composition algorithm, the average Web service composition time for different versions is 1.605 seconds and all the versions are deadlock-free. Compare with the existing algorithms, it is 0.3 seconds faster and deadlock-free guaranteed. According to the acceptance test results, the correctness of our algorithm is good. All the test cases are passed. Moreover, another advantage of our algorithm is dynamic. Whenever there are new components, our algorithm can be applied to generate new version without rewriting the specification.

5. Conclusions

In the paper, we surveyed and addressed composition techniques for Web services and proposed a dynamic Web service composition algorithm which facilitates an element for improving the reliability of Web service by applying the N-version programming technique. The composed Web services are verified to be deadlock-free by Petri-Net modeling. Furthermore, we carried out a series of experiments to evaluate the correctness and performance of the proposed Web service composition algorithm.

References

[1] A. Alves and e. al. Web services business process execution language version 2.0. In *http://www.oasis-open.org/committees/documents.php*, 2006.

[2] A. Arkin, S. Askary, S. Fordin, W. Jekeli, and e. al. *Web Service Choreography Interface (WSCI) 1.0*. W3C, http://www.w3.org/TR/wsci/, 2002.

[3] P. Chan. *Best Route Finding specification*. http://www.cse.cuhk.edu.hk/pwchan/BRF.doc, 2006.

[4] P. Chan and M. Lyu. Developing aerospace applications with a reliable web services paradigm. In *Proc. of IEEE 22nd International Conference on Advanced Information Networking and Applications.*, Okinawa, Japan, 25-28 Mar. 2008.

[5] P. Chan, M. Lyu, and M. Malek. Making services fault tolerant. In *Proc. of the 3rd International Service Availability Symposium*, volume 4328, pages 43–61, Helsinki, Finland, 15-16 May 2006. Springer.

[6] P. Chan, M. Lyu, and M. Malek. Reliable web services: Methodology, experiment and modeling. In *Proc. of IEEE International Conference on Web Services*, Salt Lake City, Utah, USA, 9-13 Jul 2007.

[7] P. P. W. Chan. *Building Reliable Web Services: Methodology, Composition, Modeling and Experiment*. PhD thesis, The Chinese University of Hong Kong, Hong Kong, Dec 2007.

[8] W.-L. Dong, H. Yu, and Y.-B. Zhang. Testing bpel-based web service composition using high-level petri nets. In *Proc. of the 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC '06)*, pages 441–444, Oct. 2006.

[9] S. Jones. Toward an acceptable definition of service [service-oriented architecture]. *IEEE Transactions on Software*, 22(3):87–93, May-Jun 2005.

[10] F. Leymann. Web services flow language (wsfl 1.0). Technical report, Member IBM Academy of Technology, IBM Software Group, May 2001.

[11] M. R. Lyu, editor. *Software Fault Tolerance*. John Wiley and Sons Inc, Apr 1995.

[12] McCabe and J. Thomas. A complexity measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, Jan 1976.

[13] J. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, 1981.

[14] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round-robin. *IEEE/AMC Transactions on Networking*, 4(3):375–385, Jun 1996.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on December 30,2020 at 06:30:58 UTC from IEEE Xplore. Restrictions apply.