DYNAMIC COMPOSITION OF WEB COMPONENTS USING CACHE PROXY SERVERS

Dr.E.Gajendran[1], Dr.S.Britto Raj[2], Dr.E.Mohan[3],Dr.S.Sharavanan[4],Dr.Leonard Gibson Moses[5], Dr.K.Ravikumar[6]

[1,2]Associate Professor, [3] Professor, Department of Computer Science Engineering, Mohamed Sathak A.J College of engineering, Chennai, India

[4]Professor, Department of Computer Science Engineering, Karpagam Institute of Technology, Coimbatore, India

[5]Associate Professor, Department of ECE, Kings Engineering College, Chennai, India

[6]Associate Professor, Department of CSE, RRASE College of Engineering, Chennai, India

[1]gajendrane@gmail.com, [2]sbrittoraj@gmail.com, [3]emohan1971@yahoo.com, [4]sharavanan.cse@karpagamtech.ac.in, [5]gibs.ml@gmail.com, [6]ravikumarcsephd@gmail.com

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. Web Services use Extensible Markup Language (XML) messages. In addition to the data exchange standard XML, three new standards Simple Object Access Protocol (SOAP); Web Services Description Language (WSDL); and Universal Description, Discovery and Integration (UDDI) let Web services talk to one another. 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).

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. This paper mainly concentrates on the Web service composition algorithm provided earlier. The algorithm is dynamic, deadlock free and also efficient than other algorithms. However, it has to compose the component each and every time the web service is requested. This has a serious effect when there a component is requested frequently. Hence to overcome the problem we have given a new innovative idea using the cache proxy servers combining both the static and dynamic approach making a performance in the previously given approach and the paper is going to explain the same.

Keywords: XML, WSDL, SOAP, UDDI, WSCI

2. Introduction:

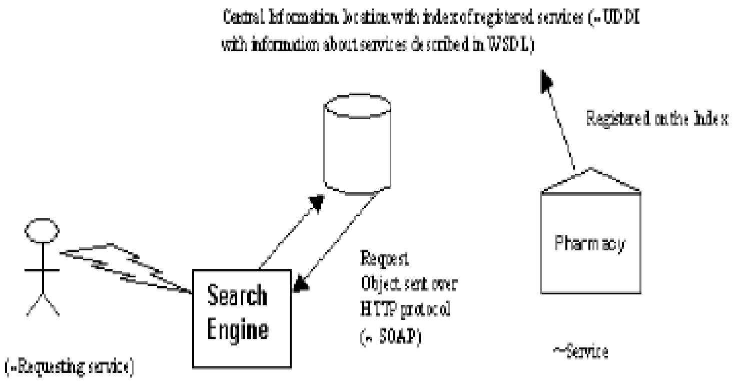
Web services, a major new trend in standards-based software technology, is made up of pieces of custom-developed code that lets two or more Web-based applications talk to each other. Instead of owning and maintaining all their own hardware and software, companies will buy IT systems as services provided over the Internet. The hope is that through the use of Web services, the blood, sweat and tears now involved in systems

integration will dissipate, leaving both companies and consumers better able to use and exchange a wide range of capabilities and information over the Internet. Web services can help companies integrate disparate systems for less money than traditional methods. Web services can meet comparable business goals can be built for tens of thousands of dollars. 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 composition approaches have been proposed for Web services in the literature. 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. Hence a dynamic Web service composition algorithm was proposed earlier which attacks these issues in a unified approach. But whenever the same composition has to be made for a frequently requested service then this is time consuming and becomes inefficient sometimes. The rest of the paper is organized as follows. Related work of Web service composition is presented in the section 3. The composition algorithm is described and the drawbacks are pointed out in the section 4. In the section 5 of this paper we describe the algorithm which is modified to serve the frequently requested services using the cache proxy servers and finally in section 6 there is a conclusion summary of the paper.

3. Related work:

The primary reason that web services are useful is that they provide a very loose coupling between an application that uses the web service and the web service itself. This allows either piece to change without negatively affecting the other, as long as the interface remains unchanged. This flexibility allows software to be built by assembling individual components into a complete application, and promises ultimate reusability of code. Web Services is based on the already existing and well-known HTTP protocol, and uses XML as the base language. This makes it a very developer-friendly service system. However, most of the technologies such as RMI, COM, and CORBA involve a whole learning curve. New technologies and languages have to be learnt to implement these services.

Also, Web Services is based on a set of standardized rules and specifications, making it more portable. This was not the case with the technologies mentioned earlier.


Central Information location with index of registered services (~UDDI with information about services described in WSDL)
Registered on the Index
Pharmacy
Request Object sent over HTTP protocol (~ SOAP)
Search Engine
(~Requesting service)
~Service

Consider a scenario in which you need to locate a particular pharmacy store in your area. You would not go out on the road and ask every person you met the way to the store. You might, instead, refer the Web site of the pharmacy on the Internet. If you knew the pharmacy's Web site, you would look it up directly and find the location through the store locator link.

If not, you would go to a search engine and type out the name of the pharmacy in the language that the search engine was meant to recognize. After getting the location, you would find the directions to the store, and then go to the store. The structure of Web Services is also very similar. Web Services provide for each of these previously described activities. If you carefully look at the preceding example, you will see that there is a requestor or a consumer—that is you. There is also a service, the pharmacy store. The central database of information is the Internet, through which you find the location of the pharmacy. In the example, when you fire a search in the search engine, your request is wrapped in a structure, whose language is predetermined and localized, and then passed onto the server running the search engine.

In Web Services, SOAP, UDDI, and WSDL represent the roles mentioned in these steps.

SOAP (Simple Object Access Protocol) is the method by which you can send messages across different modules. This is similar to how you communicate with the search engine that contains an index with the Web sites registered in the index associated with the keywords.

UDDI (Universal Description, Discovery, and Integration) is the global look up base for locating the services. In the example mentioned earlier, this is analogous to the index service for the search engine, in which all the Web sites register themselves associated with their keywords. It maintains a record of all the pharmacy store locations throughout the country.

WSDL (Web Services Definition Language) is the method through which different services are described in the UDDI. This maps to the actual search engine in our example.

WSCI (Web Service Choreography Interface) 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 characteristics as WSDL.

A number of Web service composition schemes are proposed. However, there is a need for a dynamic approach to compose the increasing number of Web services to provide new services. Hence the dynamic composition algorithm is proposed by the students of The Chinese University of Hong Kong, China which being a paper published at IEEE in 2008.

4. The Proposed Composition Method:

The 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 thecomposition 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 WSDLand 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.

```
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
```

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 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.

Case Study

To illustrate the above procedure, we present the Web services composition with the Best Route Finding system (BRF) 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.

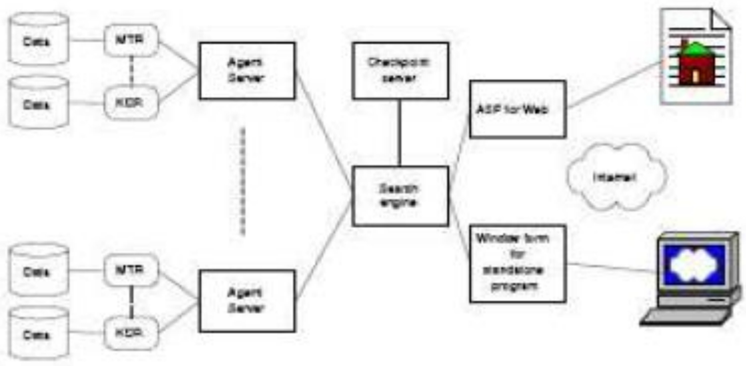


Figure 2. Best Route Finding system architecture.

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 Webservices 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>
...
```

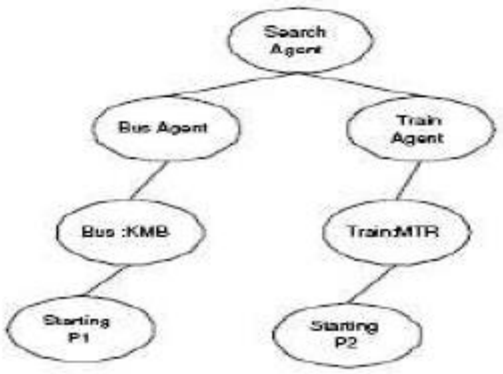


Figure 3. Composition tree of BRF.

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

The composed Web service is verified to be deadlock free by Petri-Net modeling. Furthermore a series of experiments were carried out to evaluate the correctness and performance of the proposed Web service composition algorithm.

With the proposed composition algorithm the average web service composition time for different version is 1.605 seconds and all the versions are deadlock free. Comparing with the existing algorithms it is 0.3 seconds

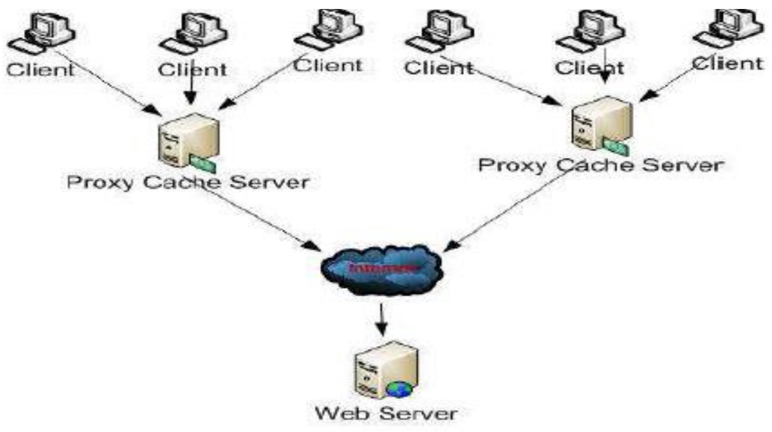
faster and deadlock free guaranteed. According to the acceptance test results, the correctness of the algorithm is good. All the test cases are passed. Moreover, another advantage of the algorithm is dynamic. Whenever there are new components, the algorithm can be applied to new version without rewriting the specification.

5. Modified algorithm using cache proxy servers:

However, the proposed algorithm is found to be dynamic and efficient, whenever single or more clients request a particular service frequently to the server, then that component has to be composed each and every time as the algorithm defines.Hence, this causes the defined time period to compose the component which is a drawback for a frequently requested service. This also needs the database access always. We modify the earlier proposed algorithm and is shown in the next page.

Proxy Servers:

We already know what cache proxy servers are. Proxy caches are often located near network gateways to reduce the bandwidth. Some times multiple proxy cache servers are used for large number of clients as the below figure 4 depicts.



Proxy cache server can be placed in front of web server to reduce the number of requests that they receive. This allows the proxy server to respond to the frequently received requests and pass the other requests to the web server. The requests passed to the web server execute the Algorithm1 as defined earlier.

In this proxy cache server, we allot a storage space for the components requested frequently. A reference for all these frequent components is maintained in a lookup table. Now if a request comes for a stored component, then there is no need to process the request to the Web server. Hence, no need to execute the algorithm. The service is directly provided through the component.

The control is transferred to the Web server from the proxy server only when the component needs to be dynamically created using the algorithm. And whatever component is newly created is stored in proxy cache server adding a reference to the table.

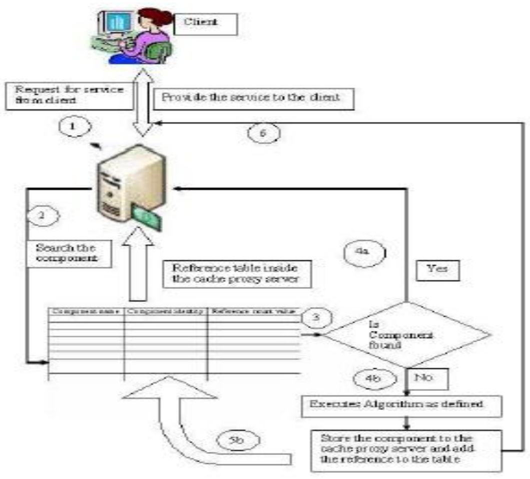
The Reference Table:

The reference table contains the component name to be referred by the client, unique component identity to search the component in the proxy cache and the reference count value to track the number of times a component requested.

Algorithm 2 Modified Algorithm for frequently requested service

```
Require: I[n]: required input, O[n]: required output
1.  CP_n: the n^th Web service component
2.  for all O[i] do
3.    Search the cache proxy server for the component CP_x.
4.    if the component found then provide the service to the client from
      the cache   proxy server.
5.    else
6.      process the Required input and output to the web server for the
        composition
7.      for all O[i] do
8.        Search the WSDL of the Web services, and find the CP_x 's
          operation output  =  O[i]. Then, insert CP_x into the tree.
9.        if the input of the operation = I[j] then
10.         Insert the input to the tree as the child of CP_x.
11.       else
12.         Search the WSCI of CP_n, WSCI process action = operation.
13.         Find the previous action needing to be invoked.
14.         Search the operation in WSDL equal to the action.
15.         if input of the operation = I[i] then
16.           Insert input to the tree as the child of CP_x
17.         else
18.           go to step (13)
19.         end if
20.       end if
21.     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 (12) to do the searching in WSCI of CP_m
22.     end for
23.   Store the new composition in the cache proxy server.
24.   end if
25. end for
```

Now you may wonder to raise a question what about the memory consumption of the proxy cache server if it keeps on storing the components newly created. This is the reason why we maintain the reference count value in the look up table. Let us explain this. Define a time period or time interval for cleaning and refreshing the table. For the defined time interval we refresh the table using the reference count. Consider for example, our time interval for refreshing the table is 30 days i.e., for every 30 days we use to clean and refresh the table and delete the unnecessary components.


Client
Request for service from client
Provide the service to the client
1
6
2
Search the component
Reference table inside the cache proxy server
4a
Yes
Component name
Component identity
Reference count value
3
Is Component found
4b
No
Executes Algorithm as defined
Store the component to the cache proxy server and add the reference to the table
5b

These unnecessary components are the components which are added to the cache proxy server but not accessed frequently after it has been added to the cache proxy server in the last 30 days. We keep a threshold value say 5 times i.e., the components accessed less than 5 times are deleted during the refreshing process. Then we reset the count value for the remaining components so that they are again incremented when accessed frequently.

6. Conclusion:

This approach combines the advantage of both the dynamic and static approaches. This composites the components dynamically and when the component is present the service is provided from the proxy cache servers which reduces the response time of the server and also reducing the network traffic. Components can easily service the clients with their quick response reducing the time delay.

7. References:

1. Chao Ma; Yanxiang He; An Approach for Visualization and Formalization of Web Service Composition

2. Joo-Yong Kim; Kyoung-Woon Cho; Kern Koh; Dept. of Comput. Sci., Seoul Nat.Univ. A proxy server structure and its cache consistency mechanism at the network bottleneck

3. Joo-Yong Kim; Kyoung-Woon Cho; Kern Koh; A proxy server structure and its cache consistency mechanism at the network bottleneck

4. Maolin Tang; Lifeng Ai; A hybrid genetic algorithm for the optimal constrained web service selection

problem in web service composition

5. Pat.P.W. Chan and Michael R. Lyu. Dynamic Composition: A New Approach in Building Reliable Web Service IEEE 2008

6. Song Wu; Hai Jin; Jie Chu; Kaiqin Fan; A novel cache scheme for cluster based streaming proxy server