AUTOMATIC COMPOSITION OF WEB SERVICES USING INTELLIGENT AGENT

By

BINOD GURUNG

Bachelor of Science in Computer Engineering

Kathmandu University

Dhulikhel, Nepal

2003

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2008

AUTOMATIC COMPOSITION OF WEB SERVICES

USING INTELLIGENT AGENT

Thesis Approved:

Dr. Johnson P. Thomas

Thesis Adviser

Dr. Venkatesh Sarangan

Dr. Xiaolin Li

Dr. A. Gordon Emslie

Dean of the Graduate College

ACKNOWLEDGMENTS

The idea of working on Web services came up during discussions with my thesis advisor Dr. Johnson P. Thomas. Later during my research, I found the idea of automatically composing Web services appealing. I would like express my gratitude to Dr. Johnson P. Thomas for his guidance and support without which this would not have been possible.

I am indebted to Dr. Venkatesh Sarangan and Dr. Xiaolin Li for willing to be in my thesis committee and helping me with questions and suggestions.

I would also like to thank all the professors from the computer science department who have contributed to this work directly or indirectly. I believe my work is heavily influenced by what they taught me.

I am very thankful for having such a caring and supporting family. Without their guidance and support none of this would have been possible.

Last, but not least I would like to thank my friends Pradeep Baral, Abhishek Ghale, Prakash Giri, Moti KC, Nabin Ghimire and Abshiek Parajuli for being there for me all the time.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

A Web service is a platform independent software system that resides in the World Wide Web and performs specific tasks on behalf of its clients. The architecture for Web services rests on principle and standards for connection, communication, description and discovery. In its most common form, clients and servers communicate using XML messages that follow the SOAP standard. The entire process is shown in figure 1.1.

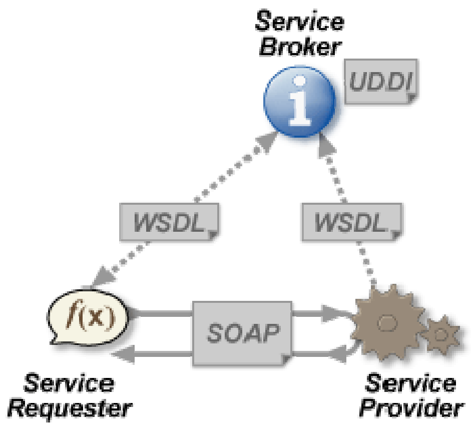


Fig 1.1 Message flow in Web service, client and UDDI

Web services these days can be simply Web APIs that can be accessed over the Internet. Examples include Payment systems by vendors like PayPal and Authoize.net. Web services advertise themselves using WSDL document which are stored by a registry system called Universal Data Description and Integration (UDDI). Clients find out about

Web services by querying the UDDI registry which returns WSDL documents. UDDI acts as both a yellow and a white page directory service [1]. Using the information in WSDL documents, client systems know how to communicate with Web services.

1.1 Composite Web service

A single Web service may not be able to fulfill a client's request. In such case, for a single task, multiple Web services have to be invoked. Hence users have to first make a list of Web services whose combination will satisfy their need. Secondly they have to locate all the required Web services using various methods like querying the UDDI registry. Thirdly they have to negotiate with different service providers and invoke those services. To simplify this task, a composite Web service can be developed which can serve as a one stop shop for its customers.

A composite Web service is a Web service that uses other Web services and performs complex tasks on behalf of its clients. Therefore, a composite Web service is invoking a sequence of other Web services rather than implementing its own service. This research is dedicated to compose an efficient Web service by using a chain data structure which can recover when one or more services in the composition fails.

1.2 Software Agent

A software agent or simply agent is a piece of code that acts on behalf of a user with authority to decide for the best action for the user [1]. These agents can learn on its own and can make decisions like changing the course of action to achieve their goal. There may be some metrics associated with agents which makes them improve themselves like humans and perform better when similar tasks are done more than once. They retain the knowledge they gain through a series of successes and failures and are capable of using that knowledge in future. The examples of Agents are many in the real world. Agents are used to sell products on the Internet with an aim to get the best price. They are also used to buy products to get the best deal. Agents use different kinds of logic like decision tree, neural network enhanced by algorithms like bagging and boosting to represent and store the knowledge. Agents can also be simple reactive agents which act based on data perceived from the environment. Such Agents uses various kinds of data structures to store knowledge. Agents are also used as Web services to provide quality and cost effective service to clients [2] and are expected to be very important players in the Semantic Web. Agents not only provide service but also collect feedbacks behind the scene and modify themselves to satisfy future customer needs. Agents are an effective way of automating a sequence of jobs that need to be done. Agents therefore are useful in automating composition of Web services and change the composition when one or more Web services fail or slow down, thereby affecting the performance of the composed Web service.

CHAPTER 2

REVIEW OF LITERATURE

For a Web service to be useful to many applications, Web services should not be too complicated [3]. When a complex Web service is needed, simple Web services can be used as building blocks. However composition is useful only when we are looking for a Web service with specific input and output parameters [3]. A lot of research has been done on different methods to compose a complex Web service using existing services [1], [3] and [1]. Most of these methods are static and some are dynamic. Static methods first collect all the component Web services and use them to compose a complex Web service. The dynamic method composes new Web services on the fly looking at input and output parameters as the request comes. These methods are robust and scalable, but since they compose Web services on the fly, they usually do not match the performance of the static services. Since a complex Web service depends on number of services, failure at one point results in failure. Therefore a composed Web service should be designed such that they can cope with such failures. Most of the research is focused on dynamically composing a Web service. Little has been done to compose a Web service with better performance than others.

2.1 Agent as Web Service

The World Wide Web is evolving into a semantic web which is a friendly environment for software agents. The web was designed for humans. Web pages were designed to be viewed by people and Hypertext Markup Language (html) is used to describe the page’s appearance. Agents however are not interested in appearance but content of the page. Semantic web is designed for humans and agents alike. Although software agents can use web as it is now e.g. shop bots, the semantic web will make the Web more accessible to agents by making use of semantic constructs like ontology represented in well established languages so that agents can understand what information the part of a web page contains.

Agents and Web services are similar in many ways. They both can also provide yellow and white page directory service to advertise their functionalities. However, agents extend Web services in many ways. Agents not only know about self unlike Web services, but they also know about the customers and their needs. It also can gain knowledge about other agents offering similar services. Such awareness can help it improve services for better customer satisfaction. Agents can be designed to use ontology which helps in better communication with clients. Agents are active and are able to provide alerts and updates when they are available. Web service standard requires it to be autonomous though it is rarely in real life. However it is a trait of an agent to be autonomous but at the same time it can form coalitions by cooperation between the agents to provide higher level and more comprehensive services. So to take advantage of these features, agents can be wrapped as web services [1].

2.2 Static and Dynamic Composition

Most Web service composition algorithms are static i.e. they compose complex Web service when all the component Web services are available. This approach has its own advantages but is hard to scale and is not robust to changes. Dynamic Web service composition is gaining ground with its obvious advantages on scalability and robustness. Pat. P.W. Chan et al. proposed an algorithm which makes use of WSCI, a XML based interface description language to describe the message flow between participating Web services [1]. It complements WSDL by describing the way operations should be choreographed. The algorithm searches the WSDL for the desired output and makes it the root of the tree. If the input of that operation is same as the required input, composition is complete. If not WSCI is searched for a matching action that needs to be invoked. This process will continue till the operation which gives the desired input is found. This algorithm also claims create a deadlock free composition [1].

2.3 Fault Tolerant Composition

Since Web services are gaining popularity, different versions of the same service are available. Thus their selection might be critical to create a fault tolerant and highly available composite Web services. Not all of these services are available all the time. This fact should be taken into consideration to compose a Web service. To compose a highly available and fault tolerant Web service, a round robin algorithm can be used to select a particular Web service among its different versions. When fault occurs, different version can be selected and UDDI registry can be adjusted to select the one which is available [1].

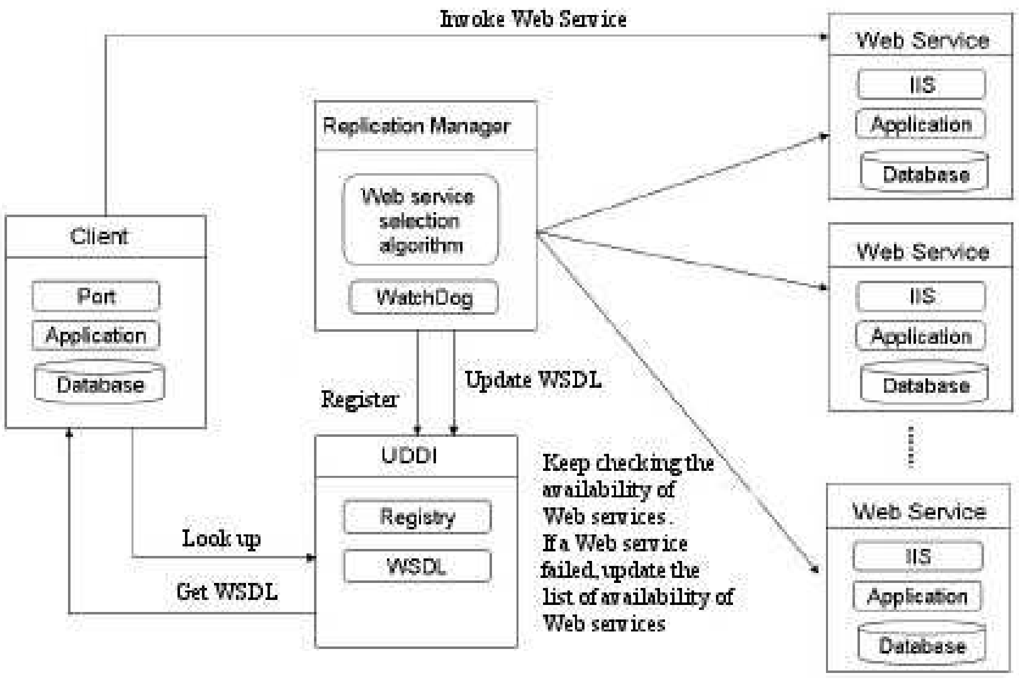
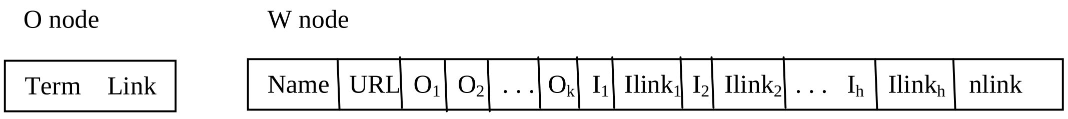
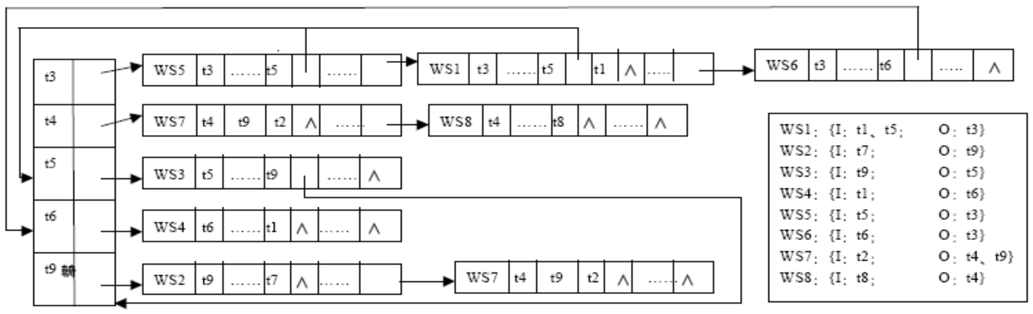


Fig. 2.1 Fault tolerant Web services [1]

2.4 Composition using chain data structure

Unlike other Web service composition algorithms, it makes use of a data structure called WS chain table which captures the dependency relationship among component Web services [3]. This algorithm is an efficient algorithm with time complexity of O (n x log (m)) where n is the number of Web services and m denotes the number of output parameters. There are two kinds of nodes in WS chain table. O node and W node. O node contains **term** and **link**. Term field contains name of an output parameter. Link field contains pointer to a W node. A W node contains name: name of Web service, URL: location of Web service, nlink: pointer to another W node, O_i: name of ith output parameter of the web service, I_i : ith input parameter of the Web service, $Ilink_i$: contains pointer to an O node whose term is same as I_i.





WS1: {I: t1、t5; O: t3}
WS2: {I: t7; O: t9}
WS3: {I: t9; O: t5}
WS4: {I: t1; O: t6}
WS5: {I: t5; O: t3}
WS6: {I: t6; O: t3}
WS7: {I: t2; O: t4、t9}
WS8: {I: t8; O: t4}

Fig. 2.2 WS Chain table and Web service inputs and outputs [3]

The algorithm maintains two lists; T which contains list of known items and G which contains list of unknowns. With current state as a root state it creates a breadth first tree with branches as Web services whose output(s) are elements of G. It updates the list of known items by adding the output of the Web service to the T list and removes it from G list. But the input to a Web service may not be in the list of known items. If that is the case, T will be updated by adding the item(s) that are required to invoke the Web service. The algorithm ends when G is empty. In the figure below, in the known items are t1 and t2 and required outputs or unknowns are t3 and t4. The composed Web service is the sequence on invocation of WS7, WS4 and WS6.

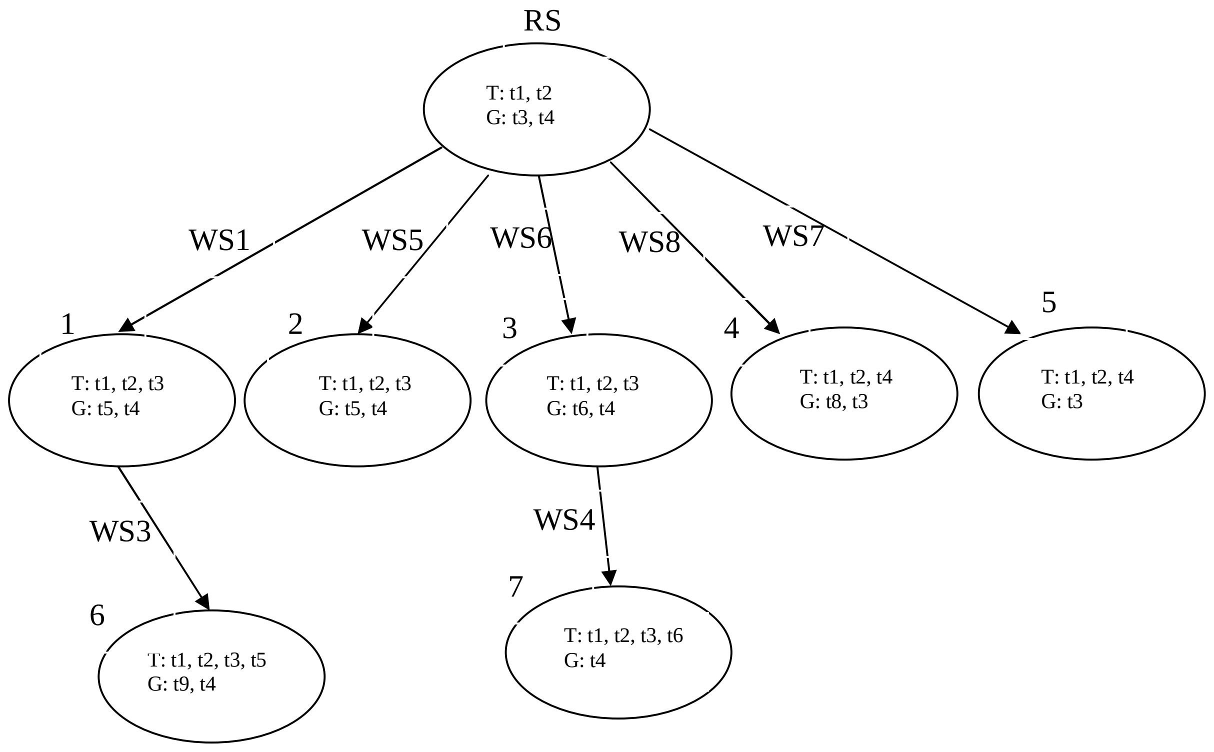


Fig. 2.3 Web service Search Tree

CHAPTER 3

PROBLEM STATEMENT

Web services are located in different machines in different part of the world. Much of the work in Web service composition has been done to address one issue at a time such as composing Web service efficiently [3], composing a reliable Web service [1] etc. When a composite Web service is created, few problems arise. Since a single composite Web service depends on a number of other Web services, probability of failure of the composite Web service increases considerably. Therefore one of the focuses of this research is to come up with a design which will make a composite Web service fault tolerant.

The other issue is the execution or response time of a composite Web service. Since there are many Web services providing same service, it is important to choose the best combination of Web services which will reduce the response or execution time. One of the techniques used is forecasting the response time of the composed service [3] and choosing the one with best forecast. This research tries to get the solution with best response time by keeping track of response times and failures of each member Web services.

CHAPTER 4

METHODS, RESULTS AND DISCUSSION

The aim of this research is to develop an agent which can compose an efficient composite Web service which can still deliver when one or more Web services in the composition fails. It uses the data structure mentioned in section 2.2 with additional fields ART (Average Response Time), FT (Full Time) and n to W node. For this research, it will be assumed that there will be one output for each Web service.

O node

Term	Link

W node

Name	URL	O	ART	TT	n	I_1	$Ilink_1$	I_2	$Ilink_2$	. . .	I_h	$Ilink_h$	nlink

Fig 4.1 O node and Modified W node

4.1 Methodology

The agent will use WS chain table to create a tree. Each branch of a tree is a composite Web service. This tree is created by using WS chain table. Each node of the tree is a W node which stores information about a particular Web service. ART in W node gives average time the Web service it represent takes to respond, n refers to number of times the Web service has been invoked, FT which is used when the W node is in tree, the approximate time it takes from that point to complete the Web service.

To create a composite Web service using the WS chain table, two things are required; the name of the inputs the composed Web service is supposed to take and the name of the output it is supposed to give. For example if we are trying to compose a Web service which takes restaurant type and zip code as input and outputs the phone number of a restaurant, the input names should be “restaurant type” and “zip code” and output should be “phone number”.

The agent will perform depth first exploration of the WS chain table to compose all possible solutions that can be achieved from the Web services whose information is stored in the WS chain table. When each solution is found, it is added to the tree as a branch. Since there will be many solutions for each composite Web service, it is supposed to be fault tolerant and selecting the best all the solutions will make the composite Web service faster. An example of composite Web service is shown in fig. 4.2 and 4.3.

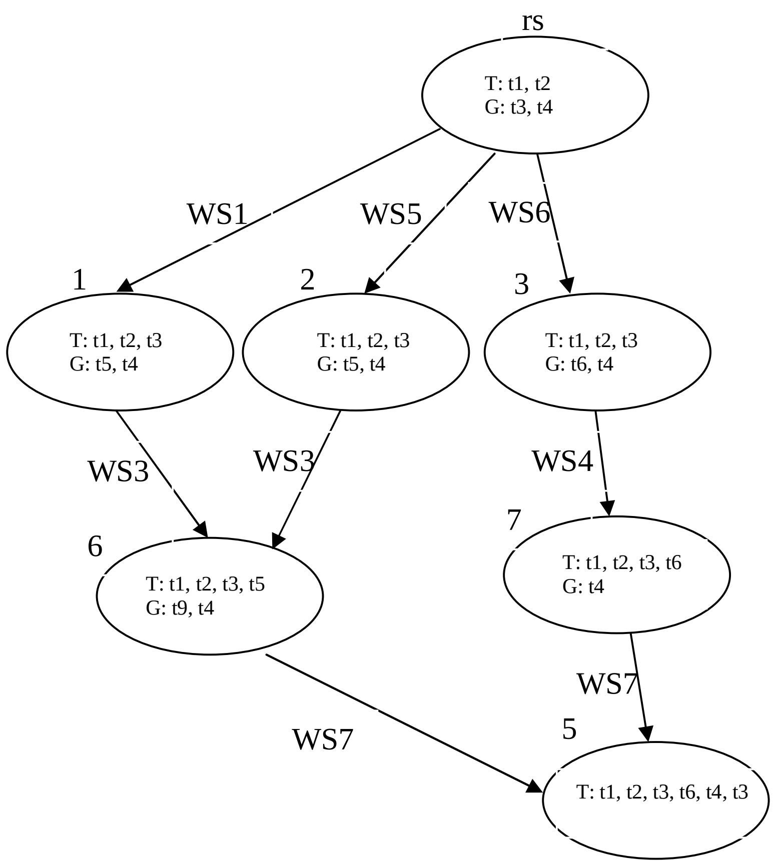


Fig. 4.2 Solution Tree in creation (logical view)

4.2 Calculation of Total Execution Time

A composite Web service is created when a tree as shown in fig. 4.2 is created. When the Web service is first used, the agent has no idea which of the combinations provides better service. So each time a composite service is invoked, the agent updates the ART value of the Web service. Using the ART value it calculates total time (TT) in each node. Fig 4.3 shows the calculation of TT by the agent. The initial value for ART and TT for each W node is 0. It means if a composite service is not tested yet, it will be treated as the best possible composite Web service and will be used next time the composite service is invoked. This way we can always be sure that the one composition we are using is really the best composition we know. The diagram below shows how TT is calculated.

Composite Service = Sequential Execution (Service 1, Service 2, Service 3)

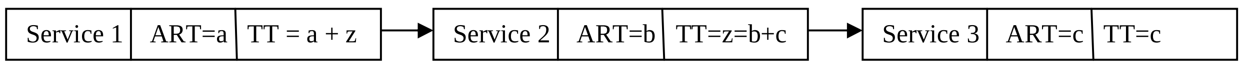


Fig. 4.3 Calculation of TT in W node

After calculation of TT in composite service, the composite services are arranged in ascending order of TT. Insertion sort is used to sort composite services. In this way, the first composite service of choice will be the one with least value of TT in the first W node. Each time a composite service is invoked, the value of ART for the component Web service is updated. The value of n in W node is also incremented by 1. The formula shown below is used to calculate new ART. The new ART will eventually be used to calculate new TT.

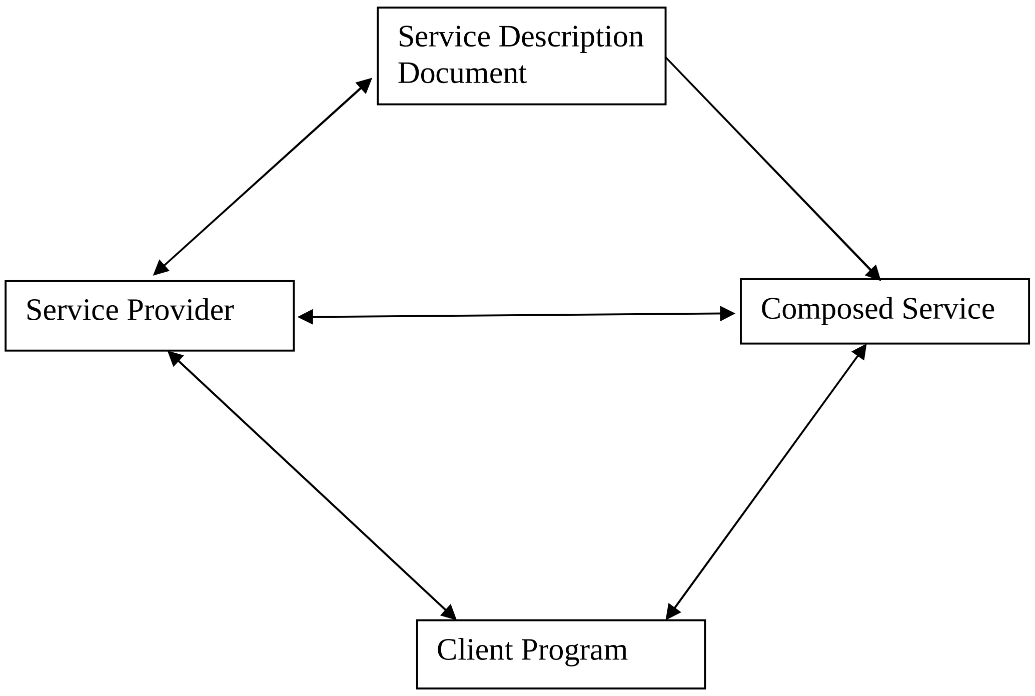
ART = (ART x n + time taken) / (n+1).

If a Web service fails, the failing Web service is penalized by updating ART value as shown below.

ART = (ART x n + 2 x ART) / (n+1)

4.3 Implementation

The agent program itself is a Web service. The user therefore first calls the agent Web service. The agent program uses a service description document which is an XML document created using a WSDL document. This document acts as a static database for the list of Web services available to use. The service description document is used to create a WS chain table which later will be used to compose a composite Web service. The client program can either invoke the Web services or can invoke the composed Web service which again uses services from the Services provider.



4.4 Experimental Setup block diagram

The Agent has two interfaces. The first interface is meant for administrators to compose Web services. To compose a Web service, the agent should know what will be the input (input names e.g. “zip code” or “food type”) and output (output names e.g.

“phone number”) to the composite Web service. When the agent creates a WS chain table and uses the input names as output names to create a Web service tree. The process is shown in fig 4.5

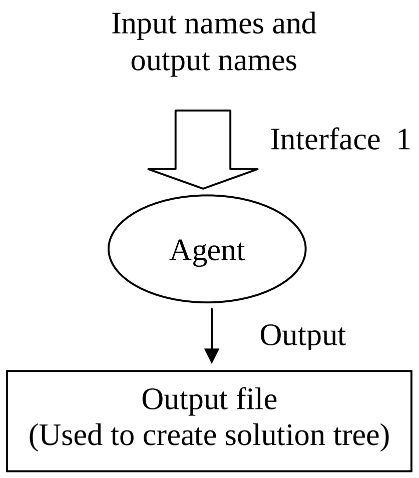


Fig. 4.5 Administrator view of the agent

The Web service tree is saved as a text file by the agent. The same tree will be used when the service is requested through the second interface. After each execution of the composed Web service, W nodes in the tree are updated and total execution time is calculated. The Web service tree is then sorted and saved for future use. The process is shown in fig 4.6.

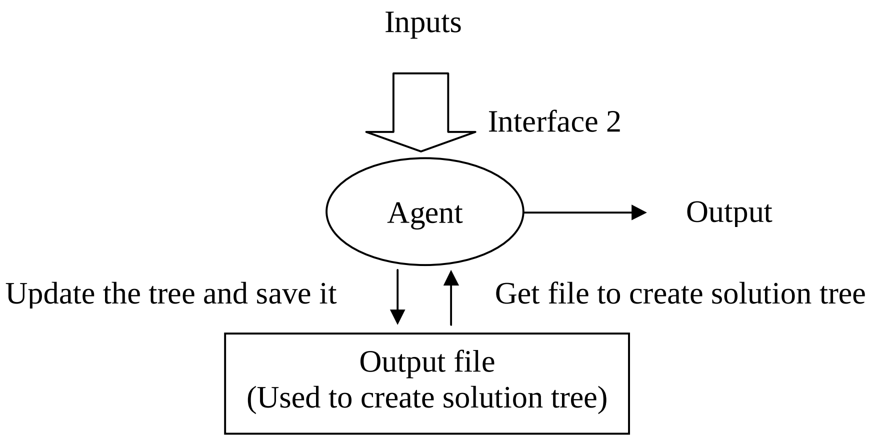

Inputs
Interface 2
Agent
Output
Update the tree and save it
Get file to create solution tree
Output file
(Used to create solution tree)

Fig. 4.6 User view of the agent

In this experiment, Apache Tomcat was used as a Web server, Apache Axis 2 was used to deploy Web services and the agent and the client program were written in Java. Sample service description document is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>

<Services>
<service>
 <name>name1</name>
 <url>http://localhost:8080/Webservice/services/WS</url>
 <inputs>
 <input>a</input>
 <input>b</input>
 </inputs>
 <output>c</output>
</service>

<service>
 <name>name2</name>
 <url> http://localhost:8080/Webservice/services/WS </url>
 <inputs>
 <input>a</input>
 <input>b</input>
 </inputs>
 <output>d</output>
</service>
.
.
.
<service>
 <name>name9</name>
 <url> http://localhost:8080/Webservice/services/WS </url>
 <inputs>
 <input>a</input>
 <input>x</input>
 </inputs>
 <output>c</output>
</service>

</Services>
```

Fig. 4.7 Service Description Document

4.4 Experimental Verification

In this experiment nine Web services are used. Each Web service is a simple Web service which takes two parameters as input and gives one value as output. The information about the Web services used is shown in the table below. The Average response shown in the table below is calculated by invoking the each Web services 50 times and taking average of the response times.

Web Service Name	Average Response Time (ms)	Error Probability	inputs	output
WS1	622	0.1	a, b	c
WS2	312	0.1	a, b	d
WS3	219	0.1	c, d	e
WS4	416	0.1	a, b	u
WS5	518	0.1	a, b	v
WS6	62	0.1	c, u	e
WS7	358	0.1	d, v	e
WS8	170	0.1	a, x	c
WS 9	313	0.1	a, b	x

Table 4.1 List of Web services used in the experiment

The inputs used in this experiment are “a” and “b” and required output is “e”. Interface 1 takes “a” and “b” as input and creates the tree shown in fig. 4.7. It finds 5 solutions as shown in table 4.2. After number of executions the based on total execution time (TT), the tree is sorted as shown in fig 4.8. Table 4.3 shows the TT for each composition after 50 executions of each composition.

Composed Service	No of runs	No of times failed when created using WS chain	No. of times failed using Agent
C1	50	10	0
C2	50	20	0
C3	50	13	0
C4	50	17	0
C5	50	10	0

Table 4.2 Composed service error chart

Table 4.2 shows the number of times each of the 5 composed Web services failed during 50 simulation runs. The right most column of the table shows that if the agent is used, it recovers from failure making the composed service fault tolerant.

Composition	Average TT
C4	1038.76
C2	1081.24
C5	1233.14
C3	1152.5
C1	1196.56

Table 4.3 Compositions and their average execution times

Table 4.3 shows the execution time of each of the composed Web services. The agent creates a tree using all the 5 composed Web services as shown in fig. 4.7. When all the 5 composed Web Services are run at least once, they are sorted as shown in figure 4.8. So when the next request comes, C4 will be executed first. If C4 fails, then the next best will be executed.

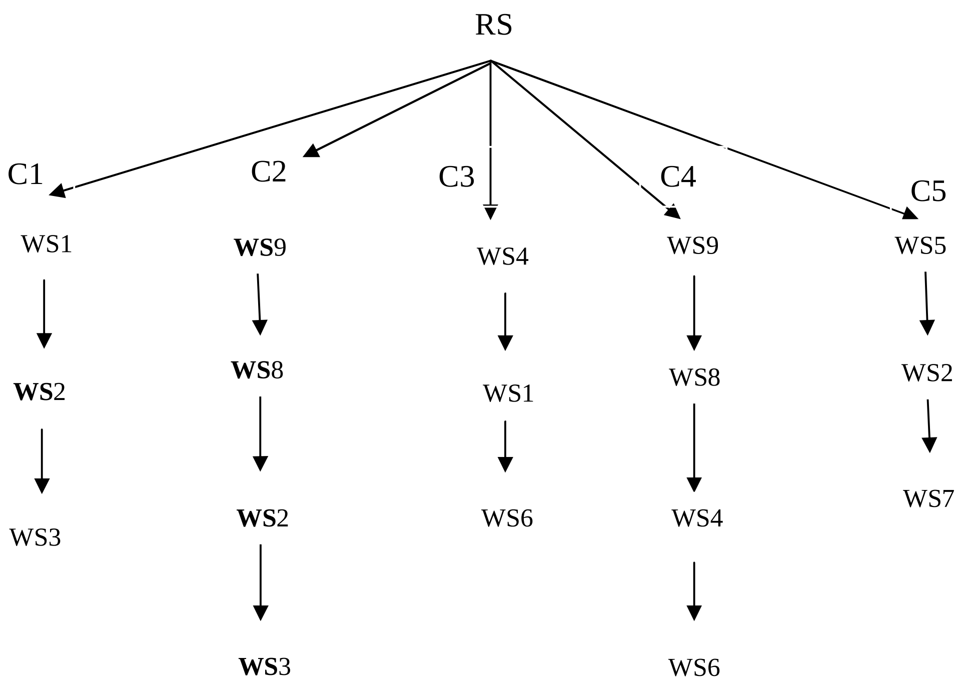


Fig. 4.8 Composed Web service Tree

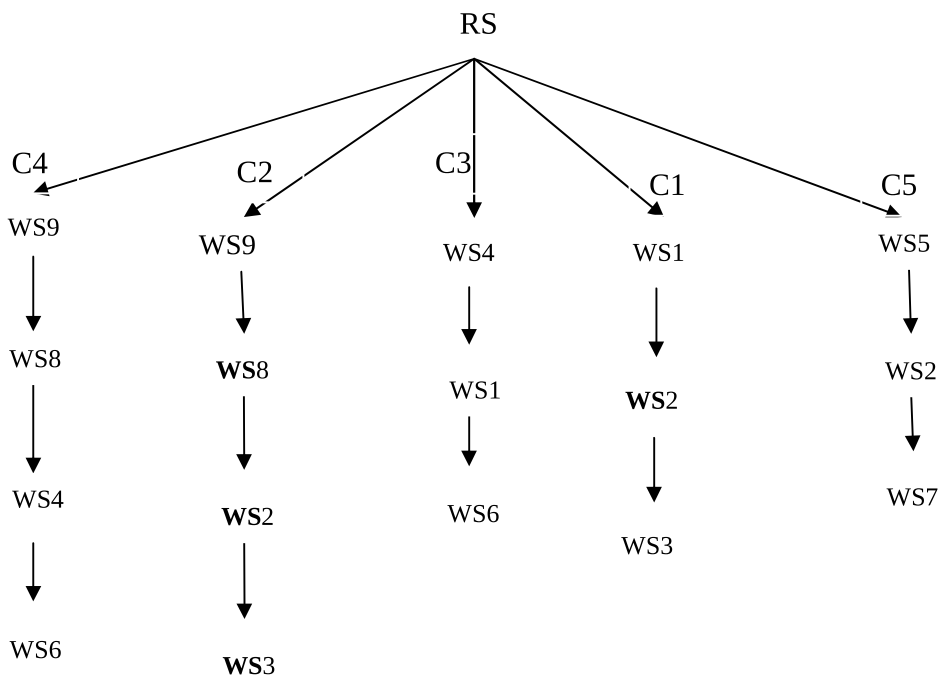


Fig. 4.9 Composed Web service Tree after sorting

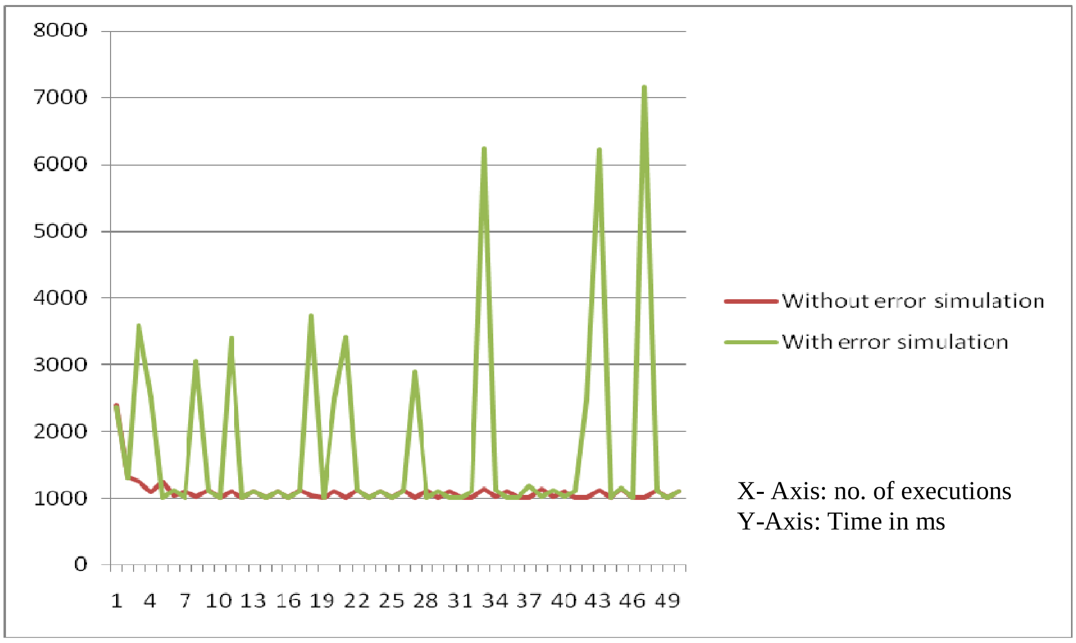


Fig. 4.10 Simulation result

The red line in the graph shows the results got with the assumption that no error occurs during execution. The results shows that after fluctuating initially, the composed Web service becomes steady which means it has found the best solution. The green line is the result when each Web service has probability of 0.1 to fail. The rise and fall seen on the graph is due to failure of one or more Web services. When a Web service fails, it moves to the next solution and hence takes more time. Apart from sharp edges, most part of the green line coincides with the red line showing that it also found the best solution and executes that solution most of the time.

It is important to note that network delays add to the response time. However, this will not affect our proposed approach as the agent measures total response time which also includes network delays. Hence, our method is applicable to local web services or remote web services.

4.5 Execution

When the agent selects one of the compositions, it executes the first Web service in composition. The agent gets the result and invokes another Web service using the result if necessary. Finally it executes the last Web service and the result is sent to the user. Fig. 4.10 shows the execution of the composed Web service. The agent first invokes WS9 and gets the result. Then it executes WS8, WS4 and WS6. The result it gets from WS6 is sent as output to the user.

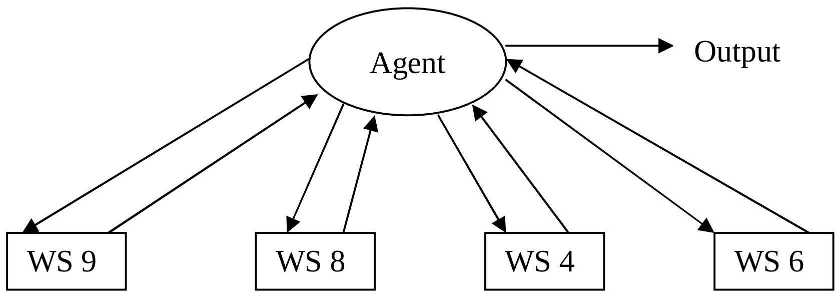


Fig 4.11 Execution of the composed Web service

CHAPTER 5

CONCLUSION

Web service composition algorithms are useful when input and output are specific. When this criterion is met we can create complex Web services using existing simple Web services. The Web services if just created may have some flaws. If one of the component Web services fails, the composed Web service fails. This way the probability of failure of the composed Web service is higher. In this thesis we propose an approach to recover when one of the composite Web services fails. We also propose an approach to determine the most efficient composed Web service out of the many different combinations of composed Web services that will fulfill our need.

Inverted chain data structure is an efficient method of storing information about Web services that we used to compose complex Web services. We use this as our database for Web services and created an agent which not only finds a single solution but finds all possible solutions to create a composed Web service which is efficient in terms of execution speed or response time and recovers from failure easily. We implemented this web service architecture and results show that our proposed approach selects the best composed service and also recovers from failure.

Many Web services in a composed Web service can be invoked in parallel. This requires finding out dependency relationship among the member Web services and agent should be able to invoke multiple Web services at the same time and process or store the results as required. This aspect of the composed Web service can be explored in future.

Some Web services can be fast but may fail often. Others can be reliable but slow. The agent can be trained to find balance between the two so that it can select a member Web service which serves out best interest.

REFERENCES

[1] P.P.W. Chan and M.R. Lyu, “Dynamic Web service Composition: A New Approach in Building Reliable Web service,” Proceedings of IEEE International Conference on Advanced Information Networking and Applications, pp: 20 – 25, 2008, [Online]. Available: http://doi. ieeecomputersociety.org/ 10.1109/AINA.2008.133

[2] J. Chi, J. Song, “Intelligent-Agent and Web-service Based service Composition for E-Business”, Canadian conference on Electrical and Computer Engineering (2007), pp: 840 – 843, 2007

[3] J. Eder, H. Pichler, “Response Time Histogram for Composite Web Services,” Proceedings of the IEEE International Conference on Web Services, pp: 832-833, 2007. [Online]. Available: http://doi.ieeecomputersociety.org/ 10.1109/ICWS.2004.1314963

[4] M. N Huhns, “Agent as Web services,” IEEE Internet Computing, vol. 6, no. 4 pp: 93-95, 2002. [online]. Available: http://doi.ieeecomputersociety.org/10.1109/MIC.2002.1020332

[5] S. Hwang, E. Lim, C. Lee and C. Chen, “On Composing a Reliable Composite Web service: A Study of Dynamic Web service Selection,” Proceedings of IEEE International Conference on Web services, pp: 184-191, 2007, [Online]. Available: http://doi.ieeecomputersociety.org/ 10.1109/ICWS.2007.133

[6] L. Li, M. Jun, C. Z. Min, and S. Ling, “An Efficient Algorithm for Web services Composition with a Chain Data Structure”`, Proceedings of IEEE Asia-Pacific Conference on Services Computing , pp. 64 – 69, 2005. [Online]. Available: http://doi.ieee.computersociety.org/ 10.1109/APSCC.2006.28

[7] S. Liu, P. Küngas, and M.Matskin, “ Agent-Based Web service Composition with JADE and JXTA”, 2006. [Online]. Available: http://www.idi.ntnu.no/~peep/papers/SWWS2006_LiKM.pdf

[8] S. A. Mcilraith, T. C. Son, and H. Zeng, Semantic Web services, Proceedings of IEEE on Intelligent Systems, pp: 43 – 56, 2001

[9] M. Ouzzani, A. Bouguettaya, “Efficient Access to Web services,” IEEE Internet Computing, vol. 8, no. 2, pp: 34-44, 2004. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/MIC.2004.1273484

[10] T. Takase, M. Tatsubori, “Efficient Web services Response Caching by Selecting Optimal Data Representation,” Proceedings of IEEE International Conference on Distributed Computing Systems, pp. 188 – 197, 2004. [Online]. Available: http://doi.ieeecomputersociety.org/ 10.1109/ICDCS.2004.1281583

APPENDIX A

DATA STRUCTURES

```
Onode

{

        Onode next;

        Term term;

        Wnode link;

}


Wnode
{
        String wsname;

        String url;

        String output;

        InputLink inputs;

        Wnode next;

        Wnode child;

        int TT=0;

        int ART=0;

        int n=0;

}
```

```
InputLink
 {
        String input;
        Onode link;
        InputLink next;
}

Term
 {
        String term;
        Term next;
}
```

VITA

Binod Gurung

Candidate for the Degree of

Master of Science

Thesis: AUTOMATIC COMPOSITION OF WEB SERVICES USING INTELLIGENT AGENT

Major Field: Computer Science

Biographical:

Personal Data: Born in Pokhara, Nepal on December 1, 1979

Education:
Received B.E. Degree from Kathmandu University, Dhulikhel, Nepal, 2003 in Computer Engineering
Completed the requirements for the Master of Science in Computer Science at Oklahoma State University, Stillwater, Oklahoma in December 2008

Experience:
Graduate Assistant in Department of CEAT Continuing Education, Oklahoma State University, 2007 to 2008

Name: Binod Gurung Date of Degree: December, 2008

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: AUTOMATIC COMPOSITION OF WEB SERVICES USING INTELLIGENT AGENT

Pages in Study: 28 Candidate for the Degree of Master of Science/Arts

Major Field: Computer Science

Scope and Method of Study:

Previous works on Web service composition focused on developing algorithms for a specific purpose like efficient composition, efficient composite service or fault tolerant composition. The aim of this research was to develop an agent which will compose efficient Web service and also recover from failure. The performance of the agent was evaluated by simulation. It was found that the proposed composed Web service performs better. It was also able to recover from failure thus reducing failure rate.

ADVISER’S APPROVAL: Dr. Johnson P. Thomas