# Self-Healing in Dynamic Web Service Composition

S.Poonguzhali[1], R.Sunitha[2], Dr.G.Aghila[3]
Department of Computer Science, School of Engineering & technology, Pondicherry University, India.
Poon_8724@yahoo.com, maanila@yahoo.com, aghilaa@yahoo.com

***Abstract*--** **Web service composition is defined as an orchestration of multiple web services into a single composite web service. Web service composition is done in three ways such as static web service composition, dynamic web service composition, semi dynamic web service composition. The dynamic web service composition is done on the fly. When the user gives the request the dynamic composer searches for suitable web services and it is composed accordingly. When the services composed dynamically they need to execute in different environment. The dynamic web service composition leads to several faults such as poor response, incorrect order, service incompatibility, unavailability etc. If the failure occurs the cause of the failure has to be detected and healed. The healing should be done automatically. Hence this paper focuses on self-healing mechanism for dynamic web service composition. Self healing is a property of autonomic computing which makes the system to heal itself from the faults. This paper focuses on QoS faults. We proposed a self-healing solution for dynamic web service composition faults. We proposed architecture for self-healing which heals the QoS faults in dynamic web service composition. We have given procedure for our healing mechanism.**

***Keywords*-- *Dynamic web service composition; self-healing; QoS failures***

## I INTRODUCTION

Web service composition is defined as the orchestration of atomic web services into a composite service. The web service composition facilitates creation of new web service from a set of available service. Thereby web service composition facilitates the reusability. There are three ways in which the web service composition is performed: static composition, semi-dynamic composition and dynamic composition. Static composition is done at design time. The dynamic composition is very challenging as it is done at run-time based on the user's request [10]. Many researches have been done on dynamic web service composition. As the component services from different service providers composed to form a composite service they may have different environment in their server and have different environment in composer side. The environments may be bandwidth, and software support. As this web services composition is done dynamically they need to balance itself when the environment changes. If the web service cannot balance itself then it leads to several faults such as ***incorrect order, misunderstood behavior***, QoS service failure such as ***poor response*** and ***service unavailability***, etc [1]. Incorrect order occurs due to message flow through SOAP. When the transfer of packets is in different order in receiver side than at the sender side it leads to incorrect order. Misunderstood behavior occurs when the requester receives the different service than he expects. This is because some times the service description may not be correct. When failure occurs the composite service will not be provided until it has been healed. QoS failures occur during run time. They are all unexpected failures. The quality of service is influenced by the parameters such as response time, availability, latency, throughput etc., which are calculated at runtime. These parameters may change time to time. This leads to deviation in quality of service. As the QoS changes, the performance also deviate from the estimated one.

The poor performance of the component service also affects the composite service. If such situation happens human intervention is needed to heal the composite service. In order heal the service the service has to stop and root causes should be detected and repaired and again the composite service has been restarted. So there is a need of self-healing mechanism in composite web service to heal itself without any human intervention and without stopping the composite service, if there is any problem occurs due to the change in QoS values.

Self-healing is defined as the process of detecting and correcting failures to system itself without any human intervention. Self-healing is a property of autonomic computing. The self-healing systems study themselves and react accordingly. Self-healing has been emerged in several software and hardware system. Many researchers have been working on self healing web service composition. The self-healing property is very helpful in dynamic web service composition as they are done on the fly. As the services combined to form a composite service at run time they may not know each other before they composed. It is the duty of composer to compose the web services in a correct flow. This is a very crucial process. So the self-healing property of web service composition studies the full composition before execution. This advanced study helps the system to detect the abnormal situation. There exist many faults in dynamic web service composition. Those failures are discussed in [1]. One of them is QoS faults. These faults occur due the deviation in QoS parameters due the changes in environment. The self-healing techniques in web service composition for QoS based failures emerged so far are ***reselect*** [9] which reselects and re-module the composite service, and ***substitute*** the replica of the original service [7]. Reselecting the service and remodeling is more time consuming because the entire composite service has to re-module. Keeping replicated services in a repository results in heavy load. If the replicated one is not available then the composite web service will go for alternative web service without considering the interrelationship between the web services. If the failure is due to the bandwidth then replacing the replicated one of the service and retrying the service are of no use. Even though we give an alternative web service without considering the interrelationships, it will give poor results. Because in a composite service, the component services communicates with each other. If any failure occurs in component service it will influence the other service also. So the alternative service should be chosen in such a way that the service should satisfy the bandwidth, input and output requirements and it also should connect with the partners in order to fulfill the healing process.

To overcome these issues we proposed a self healing approach which substitutes the alternative web service which provides the same service as that of failed service by considering the interrelationship between the component web services. The self healing is done by performance prediction. The alternative web service is replaced before the entire composite service has been stopped. The performance prediction has been done on the basis of the execution of BPEL activities which is the business activities between and within the service by means of BPEL engine.

The section 2 presents the works done related to self –healing web service composition, section 3 describes the proposed self healing technique, and we conclude our work in section 4.

## II RELATED WORKS

In this section we will review the works done on self-healing web service composition.

In [2], the self-healing policy which is an enhancement of WS-BPEL engine is proposed. This policy monitors for unexpected failures that affect the BPEL process and diagnose the root cause for the failures and find the solutions to recover from that failure. The details of the failure are stored in a database. The diagnosis part makes use of that database. If failure is not encountered in database the manual intervention is needed in this policy. In [3], a web service composition cycle called MAPE cycle is proposed. The MAPE cycle consists of three modules: Plan Generation Module in which the user request is parsed and converted into the form understandable by the composer too, Plan execution module executes the plan generated by the plan generation module. If required it replans the current plan. The Failure analysis module finds the root cause of the failure. This cycle is used to evaluate the existing self healing techniques in web service composition. In [4] authors have discussed some key technologies to achieve web service composition. They used aspect oriented programming to monitor the failures by keep tracking the WS- BPEL process. They used some statistical and machine learning techniques for diagnosis process. In recovery stage instead of giving exact solution to the failure they give desirable solutions. In [5] authors have proposed aspect based self-healing healing approach onto a three level architecture namely invitation, instantiation, and deployment they developed some exceptions for different types of failures. In [6] authors have proposed approaches for monitoring and analysis of QoS failures. The QoS parameters they considered are response time, throughput, and availability. They monitored the SOAP messages between the web services. They developed an algorithm which discriminates network and processing deficiencies. In [7] authors have proposed QoS driven approach by performance prediction. They developed availability and reliability algorithm for self healing web service composition. In [11] authors have used two special purpose languages such as WSCOL and WSREL for monitoring and recovery strategies. WSCOL is used for specifying constraints on the execution of BPEL processes. WSREL is used to state recovery strategies which should occur when the constraints are violated. They used BPEL activities for monitoring. The monitoring and recovery strategies are based on JBOSS rule engine. The recovery strategies are based on JBOSS production rule which is based on ECA (Event-Condition-Action). In [12] authors have proposed recovery framework for composite web services. This framework focuses on observable faults and interaction faults. They used interaction pattern and control pattern for monitoring. They make use of symptom database for determining the symptoms of error. Fault taxonomy is used to find the root cause of error. Formal concept analysis is used to find where to adjust the composition. Table 1 shows self-healing approaches in each work. The works that evolved so far for self healing contains self healing approaches such as reselect the service, and substitute the replicated service. Our approach selects an alternative web service which provides the same function as the failed one and it takes care of input and output of the service and the communication links between the services. Table2 shows the comparative study of existing works with proposed work.

TABLE 1. SELF-HEALING APPROACHES IN VARIOUS RESEARCH WORKS

| **Self-healing approaches** | **Monitoring phase** | **Diagnosis phase** | **Recovery phase** |
|---|---|---|---|
| Sattanathan Subramanian [2] | BPEL workflow | Symptom database | Recovery database |
| Guoquan Wu [4] | Aspect-oriented programming on BPEL workflow | Statistical learning theory | Event-Condition-Action(ECA) |
| Riadh Ben Halima [6] | QoS monitoring | Historical values of QoS | Nil |
| Yu Dai [7] | QoS monitoring | Semi markov model | Pre-backup service & alternate service without partner link |
| L. Baresi [11] | BPEL activity | JBOSS rules | Event-Condition-Action |
| K.S.May Chan[12] | Pattern, symptom database | Fault taxonomy | Formal concept analysis |
| Proposed Architecture | QoS monitoring based on BPEL activity | Based average value of QoS parameters | Provides alternate service with the consideration of partner links |

TABLE 2. COMPARISON OF EXISTING AND PROPOSED WORKS

| **Self-healing approaches** | **QoS parameters** | **Recovery** | **Parameter compatability** | **Sevice compatability** | **Time consuming** | **Heavy load** |
|---|---|---|---|---|---|---|
| Yu Dai, [7] | Response time, cost | Pre-backup | no | no | no | yes |
| Gerardo Canfora, [9] | Not specified | reselect | yes | yes | yes | no |
| Proposed work | Response time | Selecting alternate service | yes | yes | no | no |

## III PROPOSED WORK

The dynamic web service composition is done on the fly which leads to deviation in QoS of composite web service. The deviation in QoS leads to poor performance of the composite web service. In order to increase performance of the composite web service the performance of the each component service in the composite web service has to be evaluated. The self healing approach we developed is for detecting QoS based faults in a composite web service by monitoring the performance of the each component web service by evaluating the QoS parameters of the each component service. We took response time as QoS parameter. The response time value is computed at runtime when each process in the BPEL is executed. This approach has three phases such as monitor, diagnosis, recovery.

Monitor phase monitor the web services for performance. We followed event based monitoring which keep tracks of BPEL processes of the web services. In monitoring phase we have two components such as BPEL process scanner and performance evaluator. BPEL process scanner in Fig.2 detects which process is executed for which service with the help of BPEL engine. When BPEL engine starts its process execution, the BPEL process scanner detects the process for each web service and calculates the execution time of that process and reports to performance evaluator. The performance evaluator computes the response time of each web service and compares the current value to the stored value and if it is greater than the average value of previous histories of response time then it reports to the service extractor component of diagnosis part. The performance evaluator stores these response time values in a database.

The service extractor extracts the service from the composite web service and finds the partner links of that web service and stores it in a database and reports to the fault diagnoser. The fault diagnoser scans the input and output messages comes in and out of the faulty service , and it examines about the requirements of that service to orchestrate with the composite web service and sends the report to the alternate service finder of recovery phase.

The alternate Service finder finds an alternate service which suits the requirements and sends it to the service integrator. When finding the alternative web service the interrelationships between the web services also considered. It makes use of the data stored in the database maintained by diagnosis phase. The replacement is done before the execution of failed service. We have limited our work in such a way that only one service fail at a time. Fig2 shows the architecture of the system.

## WORKING COMPONENTS

When the user gives the query, the dynamic web service composer searches for the required web services which match the user requirements. After it gets the suitable services and verify whether those are composable with each other. Then it composes web services into a composite web services. When composing the web services at run time the BPEL processing and their partnership are automatically generated. After the composition all the services in a composite web services work together by means of its BPEL processes in BPEL file. The BPEL processes are invoked by the engine called BPEL engine. Each process in BPEL file of each service such as <invoke>, <receive>, <response>, <pick> etc., takes certain amount of time to execute.

This execution time is basis for the computation of response time of the web service. This response time is calculated whenever service has been invoked. The current response time is compared with average value of the stored value. If the current value is greater than average value then it is considered as there is a fault with that component service. Usually web service respond poorly due to network capacity. I.e. The composite service cannot be hold with the network capacity; the bandwidth goes beyond the limit. When the fault happens the proposed system chooses an alternative service which compensates the composite web service. This replacement should happen before the failure happens. The components involved in the proposed work as follows:

BPEL Monitor: The BPEL monitor monitors the BPEL processes of each web service whether the BPEL runtime invokes BPEL activity and how much time the web service takes to execute its process. The BPEL monitor has two components: BPEL process scanner and performance evaluator.

*BPEL Process scanner:*. The BPEL runtime of the BPEL engine executes the compiled BPEL process of each service. It has the logic of when the instance should be created determines which instance for which service is invoked. The invocation of service instance is done with the help of data store. The data store consists of what are all the active instances, message routing, variables, and partner links. The BPEL engine accesses the data store with the help of data access objects. The BPEL process scanner contacts with the BPEL engine runtime to keep track the BPEL process instances. When the BPEL engine runtime creates the new instance of a service the BPEL monitor creates the instance for the service. Whenever there is message invocation in BPEL runtime the BPEL process scanner starts it counter and it continues until the message invocation has been finished. This process is repeated for each activity in BPEL engine such as pick, receive, response etc. After it computes the time for each process and sends the time taken to execute each activity to performance evaluator. It also monitors whether the particular activity has been invoked or not. If the particular process of the particular web service instance is not invoked then it reports to the performance evaluator.

*Performance Evaluator:* The performance evaluator gets the execution values of each activities from BPEL process scanner and add those values as computation time and assign the time to execute the <wait> activity and waiting time of other activities as waiting time and calculates the response time of the service as sum of computation time and waiting time and store it in the database. This response time is calculated periodically. Whenever a new value has been computed, it compares the current value to the average value of previous response time values. If it is greater than the average value of the previously computed response time then it is considered as there is deviation in QoS values which leads to poor performance and unavailability of the service. If such situation happens, the performance evaluator tells the service extractor that the particular web service deviated from its performance.

Service Extractor: The service extractor traces the WSDL of the service which gives poor performance and extracts the requirements such as input type, output type, operations etc, and parses the BPEL and extracts the partner links, its role with the partner to make the web service fit in the composite web service. It stores all the information that is retrieved from the WSDL and BPEL in the database. The requirements may also include from the SLA. The SLA is the acronym for the Service level Agreement. The service Level Agreement is the contract between the provider and service consumer. The SLA contains the cost of the service, service availability for access, the maximum bandwidth it can provide etc. Some times violations in the SLA agreement is also lead to poor performance. The data which are collected by the service extractor is sent to the fault diagnoser to make the decisions to improve the performance of the alternative service.

Fault Diagnoser**:** The fault diagnoser checks what fault is there in the web service, whether it is due to the bandwidth which is the SLA violation or heavy load in composer side. Then it reports to the alternate service finder with the description of the failure such as the alternate service should be giving this much of bandwidth, this type of input and output should have, etc., and partner links and their roles of the web service to integrate with the composite web service. With the information given by fault diagnoser the alternate service can find more optimal service which suits the composite web service.

Alternate Service Finder: The alternate service finder searches in UDDI which serves the same service as that of the faulty service with the help of the WSDL description of faulty service and description in UDDI for the services related to faulty service. With the selected services the alternate service finder chooses the optimal service which satisfies the QoS requirements such as cost availability, etc. After selecting the alternative service it parses the BPEL file of the alternative service and inserts its partner links which the faulty service has. Then it checks whether the faulty service is in idle position. If it is in idle position the alternate service finder stores the final transaction before it becomes idle and discards the service then replaces this alternative service. It replaces the service by adding its link to the partner links of the services to which the faulty service is connected. It is done by updating the BPEL file of the services in the composite service.

Partner links repository: This repository consists of partner links and request-reply messages of the faulty web service. This repository serves for service extractor to store the partner details of the faulty service and for fault diagnoser to retrieve the partner links of the faulty service.

Service response time repository: This repository contains the response time values of each service in the composite web service. The response time values are stored by performance evaluator. It serves for predicting the performance of each web service.

## IV PERFORMANCE PREDICTION

The performance of each service has been predicted on the basis of response time of that service. The response time is calculated by computing the time taken to execute each BPEL activity in BPEL engine. The summation of execution time and waiting time is defined as response time of each web service. The response time of each web service has been calculated separately. Whenever the response time is calculated the values are stored. Our performance evaluator module is going to be implemented in connection with the BPEL engine. The performance is predicted by comparing the current response time value with the average of the stored response time values as shown in fig.1. The average of the stored response time values is considered as threshold value. This value is calculated periodically. The response time of service should be less than threshold value. The performance prediction mathematically given as:

S= {s1, s2, s3……..sn}

Rs[s] = ct[s] + wt[s] where S->set of services in the composite web service. ct->computing time of each operation involved in BPEL wt->waiting time of each activity.

Rt [s] $\longrightarrow$ threshold value of s; Rs [s] < Rt [s]

```
Procedure performance_prediction ()
{   For each service do
Let b[m]  ⟶ computing of the operation m;
W ⟶ waiting time of the request of the service s;
t ⟶  response time of each operation m;
t [m]:=b[m]+w[s]; T= t[m];
If T> average value
{Get cost[s], input[s], output[s], and partnerlinks [s];} Store partner links; Next ;}
```

**Fig.1.**Procedure for performance prediction

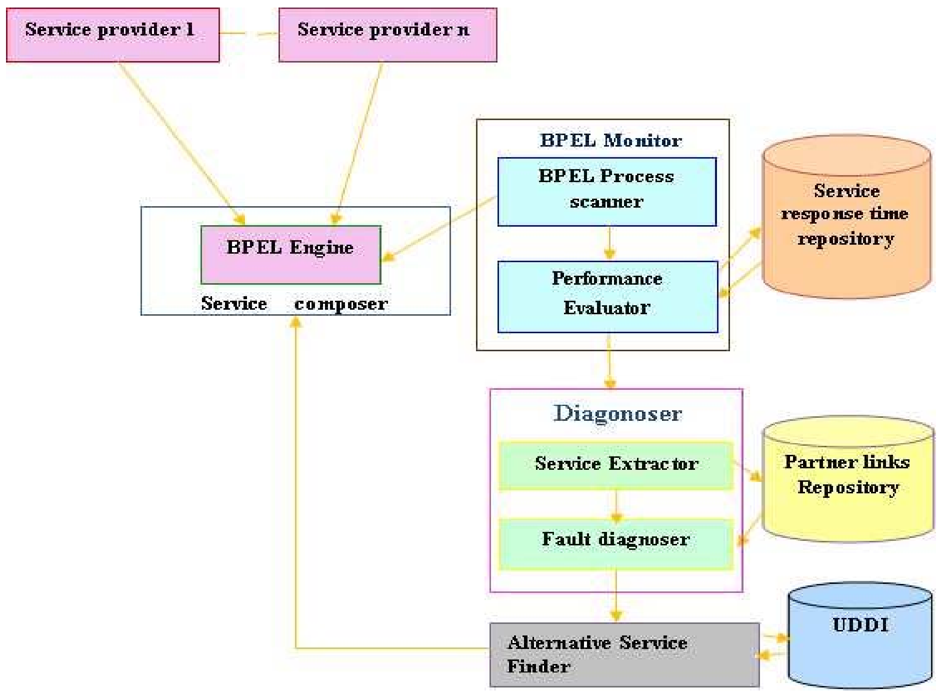

Fig.2. Architecture of proposed work

Algorithm Description for alternative service:

If the service's current response time is greater than the average value of previous value of response time then the module in alternate service finder is invoked. The steps involved in that module are given in fig.3. It gets service description, QoS and SLA requirements of faulty service and composite service in step 1 to 3. In step 4 it reads the BPEL file of the faulty service and extracts the partner links, messages and store in the repository. In step 5 it detects the current network capacity, and available bandwidth etc. In step 6 the alternate service finder searches services which matches the description of faulty service. In step 7 and 8 with the resulted services it matches the services which satisfies the QoS and SLA requirements and current network capacity and bandwidth, it chooses the more optimal service from the set of services. After getting the alternate service it parses the BPEL file of the alternate service is created with the partner links and input and output messages of the faulty service in step 9 and 10. In step 11 it checks whether the faulty service is idle. If it is idle it gets the current transaction of the faulty service and discards the service and replaces the alternative service and update the BPEL file of their partner links, other wise it waits for the faulty service until it becomes idle.

```
Procedure alternate_service () FS->faulty service;
AS->alternate service;
{
1:  get service description of faulty service
2: get QoS requirements
3:get SLA requirements
4: extract partnerlinks
5:Find network capacity
6:For (i=0; i<n;i++ do
If (desc[FS]==desc[s_i] then
S [ ]=si
End if
Next i
7:For j=0; j<n;j++ do
If COST(S[j])<=COST[FS] && Input(S[j])==Input(FS) && Output(S[j])==Output(FS) then
AS [ ]=S[J]; End if
Next j
8:For each AS={AS1,AS2,…..ASn} If(SLA requirements   is satisfiable) then
9: get BPEL description of  selected AL;
10: insert partnerlinks; End if
11: if(FS is idle) then Get last transaction Update BPEL of AL
Discard FS before it starts execution
Insert role of AL to its partnerlinks
Else
Wait for FS to become idle
Endif
}
```

Fig.3. Procedure for selecting alternate web service

## V CONCLUSIONS

Our approach to self-healing web service composition provides an alternate service for the faulty service by performance prediction. It reduces the time taken for retrying the service and reduces heavy load on the network because of keeping entire replicated composite service. We have started the implementation of our proposed architecture. We hope we will get better results.

## REFERENCES

[1] K.S. May Chan, Judith Bishop, Johan Steyn, Luciano Baresi and Sam Guinea.: Fault Taxonomy for Web Service Composition. In: ICSOC2007Workshops, LNCS 907, 363-375 (2009).

[2] Sattanathan Subramanian, Philippe Thiran, Nanjangud C. Narendra.: On the Enhancement of BPEL Engines for Self-Healing Composite Web Services. In: International Symposium on Applications and the Internet, Saint, 33-39(2008).

[3] K.S. May Chan, Judith Bishop.:The Design of a Self-healing Composition Cycle for Web Services. In: *SEAMS'09*,Vancouver, Canada, 20-29(2009)

[4] Guoquan Wu, Jun Wei, Tao Huang.: Towards self healing web service composition.In: Internetware'09, Beijing,China (2009).

[5] Ghita Kouadri Mostefaoui, Zakaria Maamar.: On Modeling and Developing Self-Healing Web Services Using Aspects, 1-4244-0614- 5/07(2007).

[6] Riadh Ben Halima, Karim Guennoun, Khalil Drira,Mohamed Jmaiel.: Non-intrusive QoS Monitoring and Analysis for Self-Healing Web Services**,** 978-1-4244-2624-9/08, 549- 554,IEEE(2008)

[7] Yu Dai, Lei Yang, Bin Zhang.: QoS-Drive Self-Healing Web Service Composition Based on Performance Prediction, J. computer science and technology, 250- 261(2009)

[8] Dmytro Rud,Andreas Schmietendorf,Reiner Dumke.: Performance Modeling of WS-BPEL- Based Web Service Composition.In: proceedings of the IEEE Services Computing Workshops,IEEE(2006)

[9] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, Maria Luisa Villani.: QoS- Aware Replanning of Composite Web Services.In: Proceedings of the IEEE International Conference on Web Services (ICWS'05), IEEE (2005)

[10] Dermian Antony D'Mello, V.S.Ananthanarayana, Supriya Salian: A review of dynamic web service composition techniques. In:CCSIT 2011,part III,CCIS 133,85-97(2011)

[11] Luciano Baresi, Sam Guinea, Liliana PasQuale.: Self-Healing BPEL Processes with Dynamo and the JBOSS rule engine.In: ESSPE'07, ACM, 11-20(2007).

[12] K.S.May Chan.: Towards a Framework for web service compositions recovery. In: ICSE- WUP'09, The warm-Up Workshop for ACM/IEEE ICSE 2010, 17-20(2009).