ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 8, August 2013

A Fault Tolerant Mechanism for Composition of Web Services Using Subset Replacement

Shuchi Gupta[1], Praveen Bhanodia[2]

Student, Computer Science & Engineering, Patel College of Science & Technology, Indore, India[1]

Assistant Professor, Computer Science & Engineering, Patel College of Science & Technology, RGPV Indore, India[2]

Abstract: Businesses offer complex services to the users, which can't be provided by a single Web Service. A Composite Web Service provides more complicated function, by composing multiple Web services. A composite service is more susceptible to failure than an atomic service. During the execution of a Composite Web Service, if one Component Service fails or becomes unavailable, the whole Composite Web Service fails. A middle agent (broker) simplifies the interaction between service providers and service requester and fulfills the user's need. The broker composes a desired value-added service and orchestrates the execution of Web Services. A replacement policy has been proposed in this paper that replaces the subset of Web Services that contains failed Web Service with another equivalent subset. During the execution, if a failure occurs, subsets containing failed Web Service are identified. Subsequently the subsets equivalent to failed one are identified. These equivalent subsets are ranked as per the policy and the best subset is selected. The old subset is replaced with the new equivalent subset in the Composite Web Service.

Keywords: Web Services, Composition, Subset Replacement, Equivalent Services.

I. INTRODUCTION

Service is a self-contained, stateless business function which accepts one or more requests and returns one or more responses through a well-defined, standard interface. Web services are self-described software entities which can be advertised, located, and used across the Internet using a set of standards such as SOAP, WSDL, and UDDI. Web services are based on Service Oriented Architecture [1, 2].

The fundamental architecture of web services is shown in figure 1.

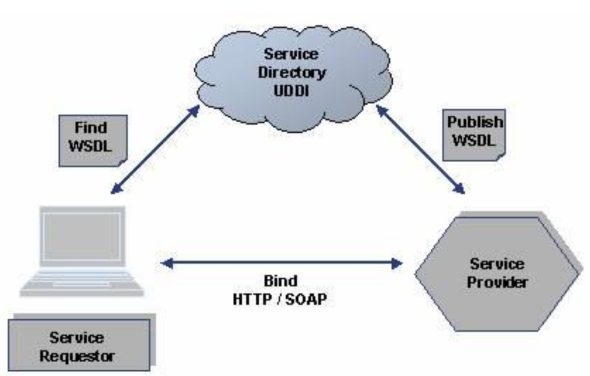

Figure 1: Web Service architechtre

A service provider creates a web service along with its definition, and then publishes the service with a UDDI. Once a web service is published, a service consumer may find the service via the UDDI interface. The UDDI registry provides the service consumer with a WSDL service description and URL pointing to the service itself. The service consumer may then use this information to directly bind to the service and invoke it.

At present, large number of Web Service are present on the World Wide Web. Most of these are designed to serve a specific type of business functionality. Present need of business enterprises are very huge in nature and can't be served by a single web service. Therefore, composition of several web services to form a complex Web service is required.

Service composition can be done either by identifying the component services in advance i.e. at the design time called Static Composition or identification at the run time called Dynamic Composition [5]. Due to the highly dynamic nature of the web, it is difficult to determine the atomic services that will constitute the composition in advance. Therefore, the composition of Web services should be done at run time, dynamically.

Execution of a composite web service includes execution of all bundled services. Thus, a composite service is more susceptible to failure than an atomic service. During the execution of a composite web service, if one component service fails, or becomes unavailable the whole composite web service fails. The business or service provider is not able to send response to the service requester. Just because of one service failure, the whole business process will not respond. In such situation the whole composite service need

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 8, August 2013

to be run completely all over again. Therefore a mechanism is needed to ensure that the running process is not interrupted and the failed service is quickly and efficiently replaced.

II. RELATED WORK

Now a day's most of the service demands from the users are answered through the web. In order to answer complex demands, composite web services had to be constructed. Senkul et al. have proposed a system that can compose web services under the constraints on the overall composite service as well as requirements on the atomic services [4]. On the basis of the constraints and acceptance levels, a set of feasible plans are generated and ranked.

There are two kinds of Web service composition, static service composition and dynamic service composition [5]. Static service composition is carried out when design or loaded, so it is less flexible than dynamic service composition, which is carried out when the application is running. Ming et al. have raised a solution for dynamic web service composition [6]. In this approach, user's requirement was broken down into a series of abstract web services. By semantic matching between the abstract services and the atomic, the executable web services composition is obtained.

Boumhamdi et al. have proposed architecture for dynamic composition of Web services as per user's requirements and availability of resources [7]. This architecture also has the ability to re-configure the composite service at runtime in case of some failure.

Execution of a composite semantic web service includes execution of all bundled services. So, a composite service is more susceptible to failure than an atomic service. Yin et al. have proposed a approach for service replacement in the composite web service. The target service could be replaced individually, or it could be replaced with its related services in the composition as a whole by another complex service. They have presented a mechanism to select the optimal service for replacement based on QoS in two phases: (1) Preliminary selection and (2) Ranking [8]. In this solution approach the best QoS service is selected for the replacement.

To address the requirements for reliable and fault tolerant Web service interactions which intercepts the execution of composite services and transparently provides recovery services. He et al. proposed an infrastructure to implement failure recovery capabilities in the Web Services Management Systems [9]. By using this infrastructure system is able to recover from the web service failure and resulting in better reliability. Erradi et al. also propose a policy driven middleware which intercepts the execution of composite services and transparently provides recovery services [10]. They define an extensible set of crucial recovery policies (e.g., retry, skip, use equivalent service), with a well-defined behavior, to declaratively specify the handling and recovery from typical faults in service-centric business processes.

Saboohi et al. have proposed a failure recovery method using sub graph replacement of web services containing a failed web service. This failure recovery method uses both forward and backward mechanisms as followings: First, re-execution of failed web service and second, execution of an alternative sub graph of web services instead of a sequence of services containing failed web service [11]. This method, composite semantic web service is considered to be a simple graph defined as S-Graph but proposed steps are of $O(n2)$ and it's most time-consuming section is the calculation of all sub graphs and finding their compatible alternatives.

Using a different approach, an exception resolving method based on discovering replacement components that are functionally equivalent is proposed by Christos et al. [12]. But this solution only replaces a single web service when a failure happens. Vaculín et al. have proposed an approach for specification of exception handling and recovery of semantic web services based on OWL-S. They have used standard fault handlers and compensation handler from WS-BPEL. By combining fault handlers, Constraint Violation handlers and standard event handlers they make possible to recover from a composite web service failure [13].

Vieira et al. presented comparison of performance and recovery in web services infrastructures in the presence of faults [14]. The approach consist a set of faults that are injected in the system and measures that characterize baseline performance (without faults), performance in the presence of faults and recovery time. Chen et al. have presented a fault detection mechanism, which is based on the queuing theory, to detect the services that fail to satisfy performance requirements. They also give a reference service model and reference architecture of fault-tolerance control centre on our fault detection mechanism [15]. But it is difficult to validate this mechanism.

H. Elfawal Mansour and T. Dillon et al. implemented a fault tolerance mechanism for component web services through rollback i.e. if there is any fault then the execution is sent back to the previous state [16]. But this paper deals with component failure.

Many papers were published during the last few years related to composition and failure recovery in web services after detail study of above papers it is concluded that many strategies were proposed for monitoring and detecting failure in web services but they did not emphasize on recovery. Moreover, many static fault tolerance strategies had been proposed in which recovery from a failure was predefined. These static strategies are not feasible enough to be used in highly dynamic web environment. In some fault tolerance schemes, if a failure is detected in Composite Service then the whole composite service is discarded and the whole Composition process is done all over again, which increases the response time. In some strategies, replacement of a single web service has been done. But in general there is a

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 8, August 2013

need to replace a sequence of web services to recover from failure in case of Composite Web Service.

III. PROPOSED SOLUTION

In this paper, a mechanism is presented for execution of a complex web service in presence of fault. A middle agent (broker) has been developed that simplifies the interaction of service providers and service requester. The proposed agent takes user's functionality requirements of the desired component services to be included in the complex service. The agent matches the parameters and find out the web services in UDDI as per user's request, constitutes a desired business process (composite web service) by composing the searched web services and then prepares them for execution of component web services in business process. During the execution of a Composite Web Service, if one Component Service fails or becomes unavailable, the whole Composite Web Service fails.

Therefore a fault tolerance mechanism is to be proposed that replaces set of web services in place of old set of web services containing failed or unavailable web service. During the execution, if a failure occurs, subsets containing failed Web Service are identified. Subsequently the subsets equivalent to failed one are identified. These equivalent subsets are ranked as per the policy and the best subset is selected. The old subset is replaced with the new equivalent subset, to complete the execution.

The architecture of the proposed system is shown in Figure 2. In the proposed approach, the complex service is offered to the users through a Broker. The proposed agent takes user's functional requirements of the desired component services to be included in the complex service and readiness to pay for the service. The user is then provided with a complete complex web service.

Following components have been included in the system:

- **Service Requester:** This component represents the actual user of the system for whom we are developing the system. The service requester enters the functional requirements of the desired component services to be included in the complex service and readiness to pay for the service.
- **Service Provider:** This component represents the actual services which are requested by the service requester. These services provide the main business functions.

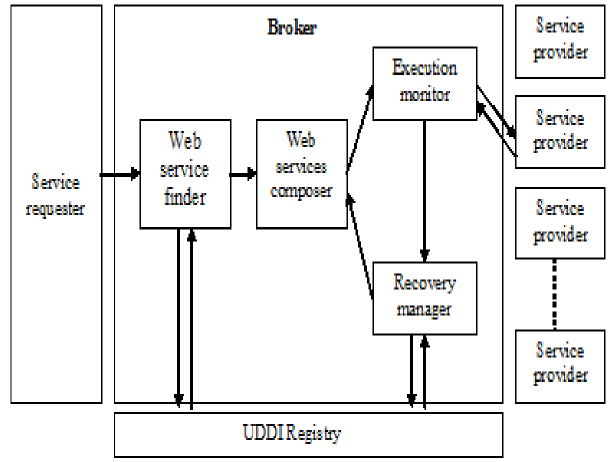

Figure 5.2: System Architecture

- **UDDI Registry:** This component contains the information of all the Web services. Then broker finds out the component service providers using in the UDDI registry to match those user's requirements.
- **Broker:** The broker contains following subcomponents.

➢ **Web Service Finder:** This component works as a searching agent according to the requirement of the user's request. It searches the service providers in the UDDI registry.

➢ **Web Service Composer:** This component composes the web services according to the user's requirement. It takes input from the web service finder and composes the component services and sends the result to web service execution monitor.

➢ **Web Service Execution Monitor:** It monitors and controls the execution of the services. If a service fails then the execution monitor detects the service which has failed and sends the information of the failed service to the recovery manager. If no failure occurs then it sends the result to the service requester.

➢ **Web Service Recovery Manager:** It find out the alternative web services in the UDDI registry for the failed web services and try to complete its execution by replacing the failed services. If recovery manager can't find a matching service node to replace, then recovery manager calculates all web service subsets in which failed services appears. Then recovery manager finds out alternatives of this web service subset and ranks them. Then select best subset according to user requirements and send it to the web service composer.

A. Fault Injection and Detection

In a computer system several faults can occur, artificial faults can be injected, and errors can be observed. To characterize a computer system in presence of faults, it is not required that the injected faults are exactly equal to real faults, it is sufficient that they cause similar behaviors (errors). What is important is to have equivalence in the

Copyright to IJARCCE

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 8, August 2013

consequence of the fault and not in the fault itself. An important aspect in fault injection is the fault trigger that describes the conditions that make the fault to be exposed (i.e., the event that leads to the injection of the fault). In the presented method, we have decided to inject faults in predefined instances in web services. We injected following fault in the system:

- **Software and development fault:** - Many time some code or variable are inserted during the development of the system and these codes are still present even though these are not required after the development and causes the faults in the system. Faults which occur due to system development or maintenance can be simulated through the code modification and code insertion technique. To inject faults in software, the part of the program is modified before the program image is loaded and executed
- **Network fault or connection loss:** In this fault, the connection between the application server and service provider server is lost. To simulate this fault we can use the timer and code insertion techniques. Using these techniques we can write the disconnection code that will activate at some fixed time after the execution starts.
- **Operational fault:** This is a kind of failure, when the operation of a service fails and we don't get the appropriate result. To simulate this fault a false code in the service operation or in the function is inserted. This faulty code is triggered or executed on a faulty condition.
- **Unexpected server crash:** In this situation server will automatically crash at run time due to functional or hardware failure. To simulate this fault the timer technique is used. In this technique after a fixed time interval, the service provider's server shuts down as written in the timer function.

In order to handle fault or error occur in Web services, exception handlers is to be implemented. These exception handlers associated with the each Service execution activity so that when an error occurs at that service, it terminates the execution, and the corresponding recovery code is executed. However, when an failure occur a signal show an exception, execution is terminated as soon as one signaled exception is caught, and only the handler for this specific exception is executed.

Replacement policy: The system replaces failed subset and executes the equivalent subset if and only if when the equivalent subset fulfils all these rules mentioned below.

1. The web service subset must provide the same functionality provided by the failed subset.
2. The web service subset must fallows the same user constrains (target location, supply days) which fallows by the failed web service subset.
3. The web service subset must have the equal cost to the failed subset.
4. The web service subset that has highest Qos parameter is selected to the replacement.
5. If already executed web services are present in the failed subset then the system must cancel the order of product by invoking cancel function of these web services.

B. Proposed Algorithms

In this section, the proposed algorithms are presented.

Algorithm to Identify Failed Subset

This algorithm first calculate the set difference of the composite set and the failed service then calculate the all subset of Remaining service Set after that take the Union of each subset with the failed service and stored in to the failed subset.

Input: Composite service set (CSS), failed service

Output: Failed subsets (FS)

Step 1: First we have to calculate the set difference of the composite set and the failed service.

Remaining service set(RSS) = Composite service set(CSS)- failed service

Step 2: Now we have to extract the all subsets of Remaining service Set RSS = {a, b, c, …}

Step 3: Then, first we separate the first element from RSS.

First-element like a then B = {b, c, …}.

Step 4: Now we use this recursion. The subsets of RES are the collection of subsets of B, plus the collection of subsets of B once again, but this time the first element a is added to these subset:

Subsets-Of (RSS) = Subsets-Of (B) + ({a} + Subsets-Of (B))

Step:5 Now then we take the Union of each subset of RSS with the failed service.

Failed subsets = Each Subsets-Of (RSS) U failed service

Algorithm to Identify Equivalent Subset

The Algorithm matches the functionality and constraints of failed subset and new subset and if both are matches then we stores this subset to the equivalent subset.

Input : newsubset[] // new subset

constraints //failed subset constraints like product name, supply days, supply location.

newconstraints[] //list of new subset constraints

F// list of the functionalities of failed subset

NF[] // list of the functionalities of new set

Output : ES[] // list of equivalent subset

Step 1: For i= 1 to all newsubset which has to match equivalent

Step 2: Match the functionality and constraints of the new subset and failed subset

Step 3: IF functionality and constraints matches

If(NF[i]= = F && newconstraints[i]==constraints)

Step 4: Then store the new subset into the equivalent sub set list

ES[j]==newsubset[i], j=j+1

Step 5: Else take the next new subset for matching

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 8, August 2013

i=i+1
Step 6: Repet the Step2 to Step5 until all new subset are not matched.

Algorithm for Alternative Subset Replacement

This algorithm identifies the equivalent web service and replaces them with the failed one. In this algorithm first identify the equivalent web services to the failed subset which have only one element if we find the equivalent one then we replace that with the failed one. Otherwise it takes failed subset which has two elements and finds out the equivalent subset if we find the equivalent subset then we replace that with the failed one. Otherwise repeat these step until all failed subsets are not finished or failed subset successfully replaced.

Input: FS failed subset in which failed service occur
CSS Composite service set
Output: NCS new composite service
Step 1. i=1 //First take the failed Service
Step 2. While (all sub set are not finished or failed subset successfully replaced)
Step 3. Select subset in which number of service=i
Step 4. Search for alternative service in juddi
Step 5. Make the possible subset
Step 6. Identify the equivalent subset
Step 7. If equivalent subset are not found
then i=i+1
Step 8. Else
Rank that equivalent subset based on cost and Qos
Select best subset
Replace failed subset to the equivalent subset (ES) in composite service set
Take the set difference (SD) of CSS and FS

SD = CSS-FS

Take the Union of the SD and ES
New Composite Service (NCS) = SD U ES
Step 9. End of while

IV. CONCLUSION

In this project a fault tolerance mechanism to alleviate failure of software systems consisting composite web services is presented. This method is based on subset replacement in a composite web service. The proposed method enables the user to avail the complex service that meet user's requirements. Even in the case of failure of a web service our method hides the failure by doing partial composition, in which failed web service subset is replaced by an equivalent subset with the failed one. QoS parameters availability, response time, and throughput are considered to determine the ranking of composite sets and equivalent subsets.

For validate this method we inject many fault in the system and simulate with the real time fault. Software and development fault, network fault, connection loss and server crash fault are simulated by the code insertion, code modification, timeout using timer fault injection technique.

If any fault occurs in the execution of a composite web service then instead of composing entire service again only the failed service set is replaced. By using our method whole composition process is not required to be repeated and only partial composition is done that improves the total response time nearly by 50% as evident from the result. Also If a service set fails during the execution, then the subsets are identified at dynamically. The proposed method reduces the number of subsets by half of total subsets. Thus to identify these subsets our method takes, up to 70% lesser time than the previous method.

In nutshell, the proposed method significantly improves the success rate and execution time in case of failures during execution of a composite service.

In future, other failure reasons in composite web service can be incorporated in our method. This project works only for the sequential web service composition. It can also be extended to work on the parallel loop structure and conditional composition of web services that require a non-liner replacement algorithm.

REFERENCES

[1]W3C, "Web Services Architecture," 2006; http://www.w3.org/TR/2004/NOTE-ws-arch- 20040211/.
[2] Wang Qing-Ming, Tang Yong, Zhang Zan-Bo, "Research in enterprise applications of dynamic web service composition methods and models", Second International Symposium on Electronic Commerce and Security, IEEE,2009, pp146-150
[3] "WSDL and UDDI", w3schools, Available at: http://www.w3schools.com/wsdl/wsdl _uddi.asp.
[4] Pinar Senkul, "Composite Web Service Construction by Using a Logical Formalism", Preceding of 22nd International Conference on Data Engineering Workshops (ICDEW'06), IEEE 2006, pp 56-65.
[5] Antonio Bucchiarone, Stefania Gnesi, "A Survey on Services Composition Languages and Models", International Workshop on Web Services Modeling and Testing, WS-MaTe 2006, pp 51-63.
[6] Wang Qing-Ming, Tang Yong, Zhang Zan-Bo, "Research in Enterprise Applications of Dynamic Web Service Composition Methods And Models", Preceding of Second International Symposium on Electronic Commerce and Security, IEEE 2009, pp 146-150.
[7] Kaouthar Boumhamdi, Zahir Jarir, "Yet Another Approach for Dynamic Web Service Composition", International Conference forInternet Technology and Secured Transactions, 2009. ICITST 2009.
[8] Keting Yin, Bo Zhou, Shuai Zhang, Bin Xu, Yixi Chen, "QoS-aware Services Replacement of Web Service Composition" 2009 International Conference on Information Technology and Computer Science pp271-274.
[9] Weiping He, Virginia Tech, "Recovery in Web Service Applications", International Conference on e-Technology, e-Commerce and e-Service (EEE'04), IEEE, 2004.
[10]Abdelkarim Erradi, Piyush Maheshwari, Vladimir Tosic, "Recovery Policies for Enhancing Web Services Reliability", International Conference on Web Services (ICWS'06), IEEE, 2006.
[11]Hadi Saboohi, Amineh Amini, Hassan Abolhassani, "Failure Recovery of Composite Semantic Web Services using Subgraph Replacement" Proceedings of the International Conference on Computer and Communication Engineering 2008 May 13-15, 2008 Kuala Lumpur, Malaysia, pp489-493.

[12] K. Christos, V. Costas, G. Panayiotis, "Towards Dynamic, Relevance-Driven Exception Resolution in Composite Web Services", 4th Int. Workshop on SOA & Web Services

[13] Roman Vaculín, Kevin Wiesner, Katia Sycara, "Exception handling and recovery of semantic web services", Fourth International Conference on Networking and Services, IEEE, 2008, pp217-222.

[14] Marco Vieira, Nuno Laranjeiro, "Comparing Web Services Performance and Recovery in the Presence of Faults", International Conference on Web Services (ICWS 2007),IEEE,2007.

.

[15] Hao-Peng Chen, Cheng Zhang, "A Queueing-Theory-Based Fault Detection Mechanism for SOA-Based Applications" The 9th IEEE International Conference on E-Commerce Technology and The 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services(CEC-EEE 2007),IEEE,2007.

[16] H. Elfawal Mansour and T. Dillon, Fellow, IEEE, "Dependability and Rollback Recovery for Composite Web Services", IEEE TRANSACTIONS ON SERVICES COMPUTING, VOL. 4, NO. 4, OCTOBER-DECEMBER 2011.