A Self-healing Model for Web Service Composition in Dynamic Environment

Aram Alsedrani[1] and Ameur Touir[2]
[1,2]College of Computer and Information Sciences, Department of Computer Science, King Saud University, KSA

Abstract– **Web service composition development is a complex and dynamic process. It is one of the challenges in distributed environments. Nowadays, the web environment becomes more dynamic due to the increase in the number of web services that are changing frequently. Therefore, the need for adaptive composition methods that act according to environment changes is advocated. Moreover, detecting changes and acting accordingly in the early stages of the composition will ensure the reliability and prevent, in some cases, the expensive re-planning process. We aim in this paper at proposing a web service composition model based on service clustering to react with service failure occurred during any phase of composition process.**

Keywords– **Web Services, Service Composition, Service Clustering and Self-healing Composition**

I. INTRODUCTION

Since the emerging of web service paradigm, it has been widely used and gains users acceptance, which encourages many developers to deploy their applications as web services [1]. This success, typically, leads to the growth in the number of available web services over the net and thus increasing the dynamicity of web service environment. In other hand, web service composition process that is based on integrating primitive web services into one complex service must rely on reliable and valid web services [2]. Web services in dynamic environment are updated on-the-fly which threatens the success of service composition process. Therefore, advanced approaches that react to changes in web service environment to insure the accuracy of the composition process becomes of highly demands. In this paper, we introduce the self-healing model for web service composition that monitors the composition process from early stages in order to recover service fails directly when it occur.

Most existed service composition methods handles the dynamism of the web environment in the execution phase and treats the environment as static in early phases. In such situations, any change will be detected and handled after their occurrence. This mechanism causes undesirable problems at execution time that leads to decrease the composition performance with re-planning or re-selecting overhead. As example of these researches is the recovery mechanism introduced in [3] and [4] that is based on validating the services in the composition in execution phase and replacing the faulty services by a sub-diagraph with the same constraints. Moreover, the work by [5] presents service composition framework based on performance prediction. At runtime, when a service is predicted to be failed according to its quality values, a re-selection algorithm will be triggered to replace the failed service before it invoked. Another example is the research by [6] that utilizes the ECA rules in multimedia conference systems to manages web service composition in the case of updating user requirement. The proper event will trigger when the business process request is changed and allow service rescheduling.

We aim in this paper at proposing a web service composition model based on service clustering to react with service failure occurred during any phase of composition process. The proposed model consists of two parts: the service clustering components that group the similar web services based on their precondition and effect into one task cluster in order to recover the failed web service with equivalent one from its cluster, and groups the similar tasks in to one job cluster. The second is the composition engine component that is monitored during its entire process in order to detect changes and react accordingly by searching for a substitution to the fault service.

The rest of the paper is organized as follow, in section two; we provide background information on the service composition and the OWL-S type of service description uses in our model. In section 3, we propose the self-healing model including the lustering mechanism and the composition engine, and then the system model components are described later in the same section. We conclude our paper in section four.

II. BACKGROUND

A. *Web Service Composition*

Web service composition is the process of combining several services into one service with upgraded functionality [7]. Generally, the composition lifecycle consists of four phases as shown in figure 1. Planning is the first phase, which aims at decomposing the requested service based on the user inputs into an abstract set of tasks known as the abstract plan[8]. Then, the service discovery is the process of searching for candidate web services to fulfill the tasks in the abstract plan. The discovered services for each task are equivalent in their functionality but differ in the non-functional requirements [9].

Moreover, The third phase is the selection which is the process of selecting the optimal web services from the set of discovered candidate services based on a set of requirements

[10]. The optimality of a web service is determined based on its QoS values as well as the user preferences. The last phase of service composition is the execution of the recently created execution plan. The process involves invoking services, passing data between participant services and verifying the composed service [11].

B. OWL-S

There are many standards to describe web services as WSDL and OWL. In our model we are building the composition plan based on semantically matching the precondition of a web service by the effect of another. For that reason, a semantic web service description is needed. Some of description standards as WSDL list the syntactic description of a service; others describe the web service semantically as OWL and OWL-S.

OWL-S is build based on Web Ontology Language (OWL). The motivation for defining OWL-S is the use of ontologies to describe web service allowing services to be machine-interpretable. Thus, it enables the automation of Web service discovery, web service invocation, and web service composition [12].

The service ontology provides three type of service. First is the service profile that describes what the service dose, it contains tags as pre-conditions, result, inputs and outputs. Second is the service grounding which describes how to access the service. Third is the service model that describes how to use the service by detailing the semantic content of requests and condition must be validate to reach service outcomes [13].

III. SELF-HEALING WEB SERVICE COMPOSITION MODEL

In this section, we present the clustering based model for web service composition. The model consists of two parts. First, an offline part that classifies the web services based on their constraints (precondition and effect) into task clusters, consequently, groups the similar tasks into job clusters. The second is an online part that receives the requested service and user inputs then builds the composed service. The composition process in our model is monitored through its all phases to detect any failure that could threaten the construction of the composed service.

A. Service Clustering

Service clustering is the process of classifying the web services that stored in the service repository into tasks based on their constraints. And then group the functionally similar tasks into jobs. In this mechanism we maintain two types of repositories: task repository and job repository.

To demonstrate the service clustering, assume the constraints dependencies in Fig. 2(a) graph denoted as G(V,E). Whereas V is the set of constraints between web services, and E is the set of tasks that leads the precondition vertices to the effect vertices. In another word, the expression T1 (u, v) that is extracted from the graph branch “u -- T1 -- > v “denotes that ‘u’ is the precondition of T1 and ‘v’ is the effect of T1.

The task clustering is the process of grouping all the similar web services in to one task. Each newly added web service to the service repository will trigger the task clustering process that will insert the new service to a corresponding task cluster. In case of no task cluster matches the service constraints, a new task cluster will be created based on the service details. The task repository as in Fig. 2(b) consists of a set of task clusters generated based on the graph of Fig. 2(a). Each task cluster is a set of web services having the same precondition

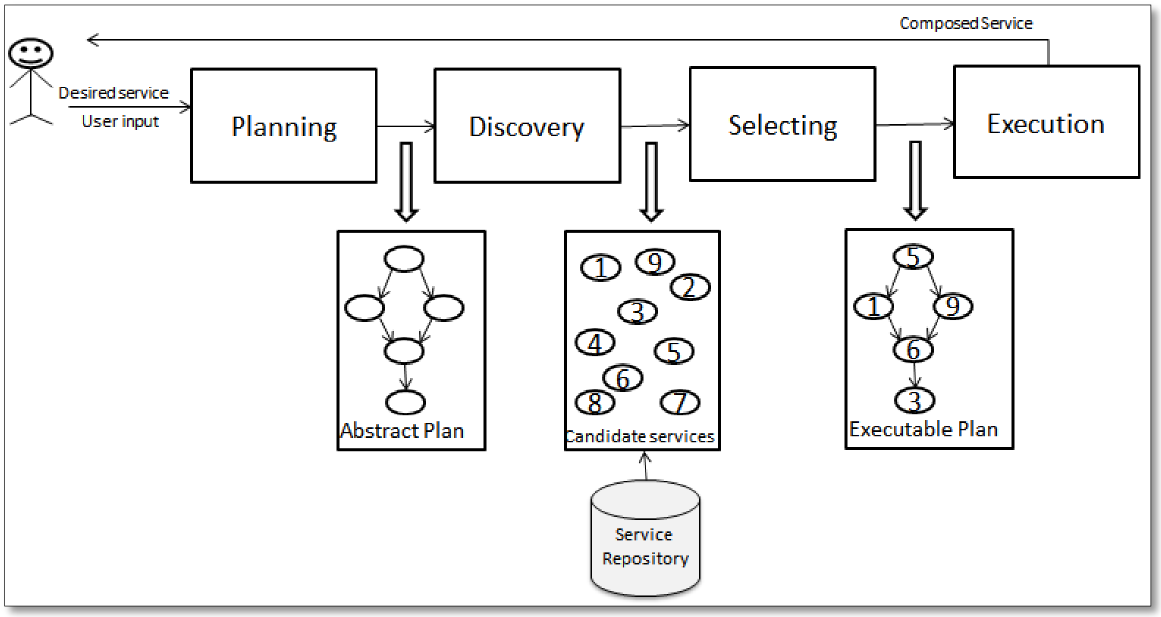


Fig. 1: Web service composition process

and result. We assume that each web service is atomic and dose not composes of other services.

In the other hand, the job repository is constructed by clustering the similar tasks in to one job. In fact, the repository is built in two levels: first, for each task in task repository a job is created in job repository, second, a composition process is performed in order to generate composite tasks for the job clusters. For each job cluster, the job precondition is inputs to a planning process in order to match the precondition with some task effect until reaching the job effect. Therefore, each job cluster consists of a single atomic task and several composite tasks. Fig. 2(c) represents an example of the job repository that is built based on the constraints of Fig. 2 (a). The expression J5(x,z) is a job with precondition of 'x' and effect of 'z', the jobs set consist of three tasks that all leads 'x' to 'z' in different paths.

The clustering is a continuous process performed whenever a new service is assigned to the service repository. Moreover, the clustering process is done in the background of the model which will not interrupt the composition process.

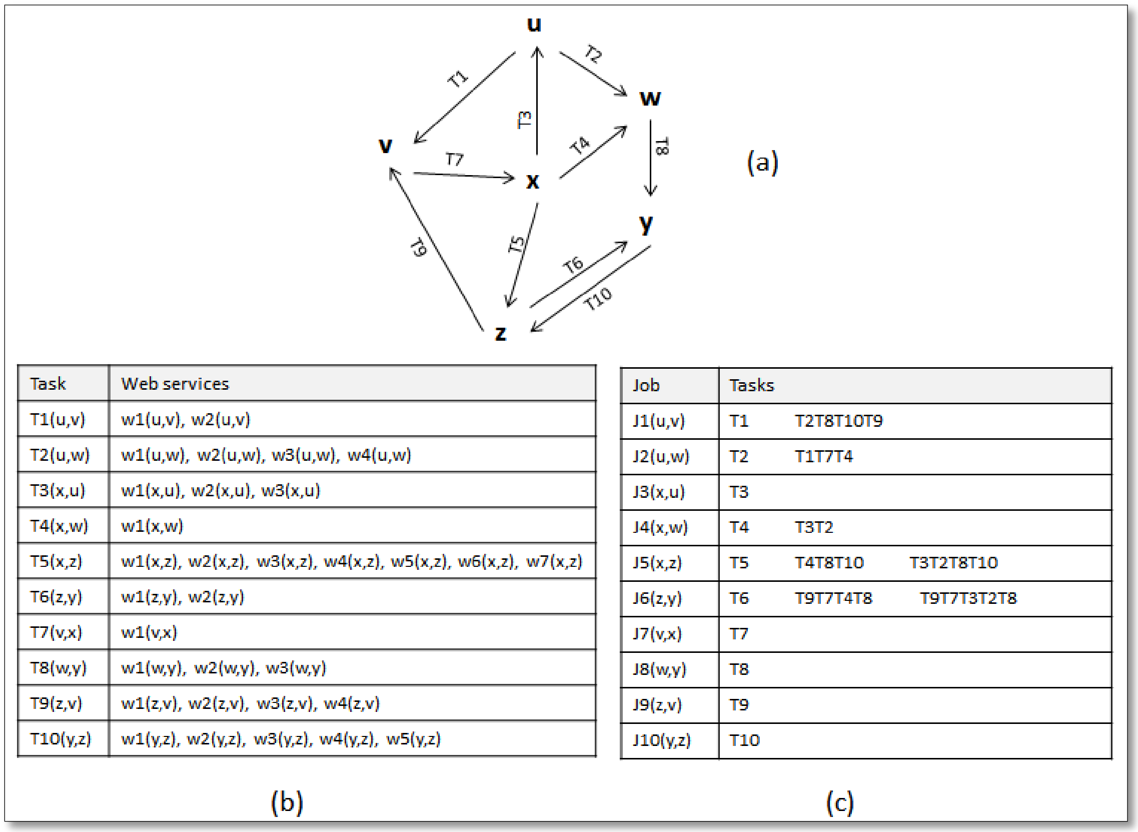
B. Clustering-based Web Service Composition

The composition model receives the user requested service and initial input required through its composition engine part as in Fig. 3. Then the planning process begins by semantically match the user input with a job preconditions. And continually, the planner matches the effect of the last job in the plan with some existed job precondition until the requested service is achieved. The outcome of the planning phase is a plan which consists of connected jobs that leads the user inputs to the user requested service.

The second phase of the composition is selecting the best web service for each job in the plan based on its QoS attributes. For each job in the composition plan, the candidate task from the corresponding job cluster is retrieved, then the candidate web service for this candidate task cluster substitute the job in the plan. The candidate web service is chosen among similar services in the same cluster if it offers the best quality attributes. Therefore the composition plan will consist of connected web services that are ready to be executed.

At last, the executer will receive the web service plan and begin to invoke each web service sequentially. The output of one service is passed by the executor to the next service as an input. The process will continue until producing the desired output of the composed service.

The web service repository in the model is monitored during the composition phases mentioned previously to detect any changes that affect the composition. In case of web service failure, the monitor component in Fig. 3 will trigger the recovery process and attempt to react and solve the failure. The recovery process is performed in two levels. The first level is based on substituting the faulty service with atomic similar one from the exact task cluster. The second is performed in case of there is no atomic alternative found, it is based on searching the job cluster to find a composite task



Task	Web services
T1(u,v)	w1(u,v), w2(u,v)
T2(u,w)	w1(u,w), w2(u,w), w3(u,w), w4(u,w)
T3(x,u)	w1(x,u), w2(x,u), w3(x,u)
T4(x,w)	w1(x,w)
T5(x,z)	w1(x,z), w2(x,z), w3(x,z), w4(x,z), w5(x,z), w6(x,z), w7(x,z)
T6(z,y)	w1(z,y), w2(z,y)
T7(v,x)	w1(v,x)
T8(w,y)	w1(w,y), w2(w,y), w3(w,y)
T9(z,v)	w1(z,v), w2(z,v), w3(z,v), w4(z,v)
T10(y,z)	w1(y,z), w2(y,z), w3(y,z), w4(y,z), w5(y,z)

Job	Tasks		
J1(u,v)	T1	T2T8T10T9	
J2(u,w)	T2	T1T7T4	
J3(x,u)	T3		
J4(x,w)	T4	T3T2	
J5(x,z)	T5	T4T8T10	T3T2T8T10
J6(z,y)	T6	T9T7T4T8	T9T7T3T2T8
J7(v,x)	T7		
J8(w,y)	T8		
J9(z,v)	T9		
J10(y,z)	T10		

Fig. 2: Service Clustering Example

that substitute the faulty service. Therefore, the recovery process acts only in the boundary of the job cluster without interfering with the rest of the jobs in the composition plan.

By monitoring the process of composition from early phases, the reacting to solve service failure, if any occur, can prevent the composition failure and reduces the time needed to recover. In another word, if a web service, that is part of the composition, fails during planning phase and detected in the same phase then, in worst case, the recovery will be the time needed for re-planning the composition. Whereas, if the fails detected in execution, as in [14] and [15] for example, the time needed for recovery will cause the re-planning and re-selecting then re-executing of the composition.

C. System Model

Fig. 3 demonstrates the clustering-based web service composition model. The model is divided, in general, in two parts: the clustering part and the composition engine part. Each component of the model will be described in this section in details.

Semantic Matcher: this component is responsible of finding a semantically matches between a precondition and an effect. The matcher could be developed using several similarity approaches. One of the approaches as example is the "WordNet-based semantic similarity measurement" [16]. The tool is a dictionary-based semantic matcher that accepts two sentences and produce a percentage that represent the degree of similarity between inputs.

Monitor: the monitor is the component that listens continually to the service repository in order to detect any changes occurred. When a change is occurred, the type of the change is sent to the corresponding recovery process of the on-going composition phase in the exact time of the fail occurrence to deal with that event.

Clustering:

OWL-S parser: that responsible of receiving the semantic web service description in OWL-S format and extracting the needed service parameters for the composition process. The main service parameters that are extracted by the parser are the service inputs, output, service constraints (precondition, effect), and service description. After that, the parser stored that information into the service repository.

Task clustering: the clustering process is implemented in the task clustering component. Each service in service repository will be clustered based on their precondition and effect parameters to specific task and stored in the task repository. The single cluster in the task repository consists of task name, precondition, effect and the set of equivalent web services.

Job clustering: this component is responsible of clustering the similar tasks into one job cluster. It generates all possible composite tasks to be added in job clusters. The job clusters will be stored in the job repository. Each entry in the job repository includes job name, precondition, effect and the set of tasks.

Composition Engine:

Planner: in which the planning process is implemented. The planner will receive the requested service (UR) and user

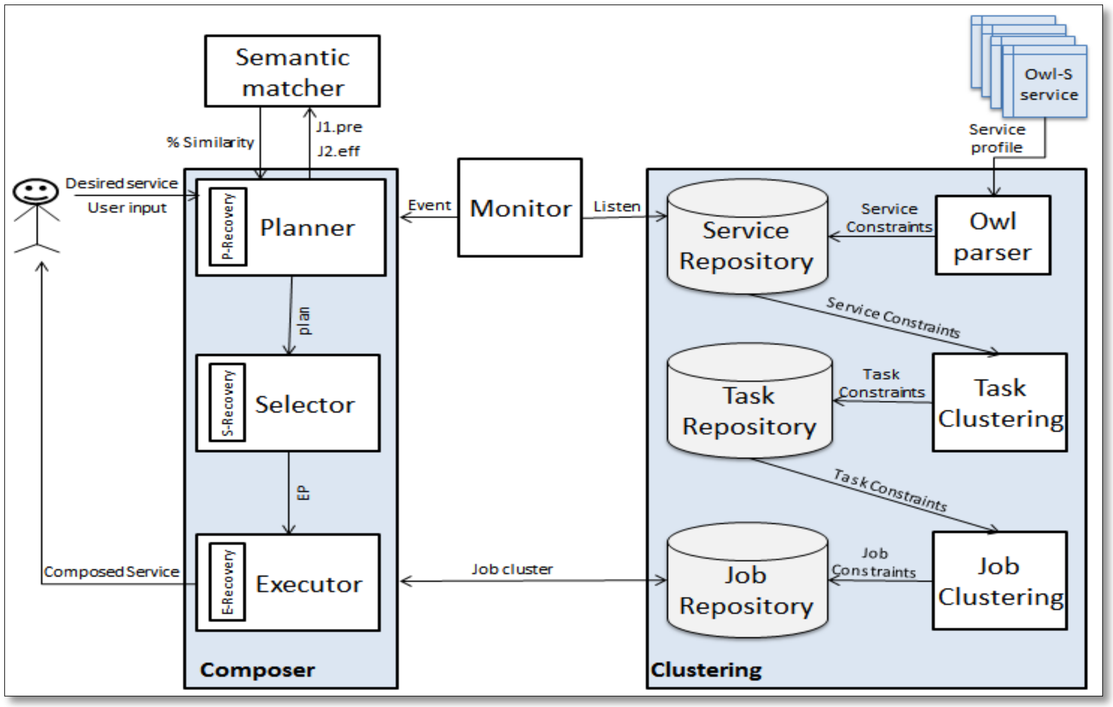


Fig. 3: Clustering-based Web Service Composition Model

inputs (UI) from the user. The planner start processing the UR by fined a match between UI and some job affect by passing those values to the semantic matcher and continually constructing the plan until reaching the UR. Whenever an event that affects planning occurred, the recovery process will be triggered to react to that event and continue planning. Finally the composition plan will be delivered to the selector as a graph of connected abstract jobs.

Selector: the selector component accepts the composition plan and substitutes each job with the candidate web service from the same cluster to construct the execution plan (EP). The selection recovery process will be triggered if the candidate web services are failed. The process searches then for an alternative service with the best quality among the remaining services in the job cluster. Then, the execution plan EP will be delivered to the executor.

Executor: the last component in the composition engine is the executor that receives the EP and delivers the composite service output to the user. The executor invokes the web services in the plan starting from the first service and continually passes the output from one services as input for the next to be invoked. If one service within the EP fails before invoking it, the execution recovery process will be activated to substitute the faulty service from the task cluster.

IV. CONCLUSION

In this paper, we present a web service composition model based on clustering the similar available web services in service repository into tasks, and cluster the similar tasks into jobs. By proposing the service clustering technique, we aim at three goals: First, minimizing the composition problem state space within jobs rather than services. Secondly, the ability of fast recovery in case of service failure by substituting the faulty service from the corresponding cluster. Third, isolate the faulty web service in its job boundary and not affecting the rest of the well composed service.

REFERENCES

[1] K. Wiesner, R. Vacul, M. Kollingbaum, and K. Sycara, “Recovery Mechanisms for Semantic Web Services,” in *International Conference on Distributed applications and interoperable systems (DAIS)*, 2009, pp. 100–105.

[2] N. H. Rostami, E. Kheirkhah, and M. Jalali, “Web Services Composition Methods And Techniques: A Review,” *Int. J. Comput. Sci. Eng. Inf. Technol.*, vol. 3, no. 6, pp. 15–29, 2013.

[3] H. Saboohi and S. Abdul Kareem, “An automatic subdigraph renovation plan for failure recovery of composite semantic Web services,” *Front. Comput. Sci.*, vol. 7, no. 6, pp. 894–913, 2013.

[4] S. Gupta and P. Bhanodia, “A Flexible and Dynamic Failure Recovery Mechanism for Composite Web Services Using Subset Replacement,” *Int. J. Sci. Res.*, vol. 3, no. 12, pp. 1886–1890, 2014.

[5] Y. Dai, L. Yang, and B. Zhang, “QoS-Driven Self-Healing Web Service Composition Based on ã,” *J. Comput. Sci. Technol.*, vol. 24, no. 2, pp. 250–261, 2009.

[6] Z. Ying, C. Junliang, C. Bo, and Z. Yang, “Using ECA rules to manage web service composition for multimedia conference system,” in *2nd IEEE International Conference on Broadband Network & Multimedia Technology*, 2009, pp. 545–549.

[7] Q. Z. Sheng, X. Qiao, A. V Vasilakos, C. Szabo, S. Bourne, and X. Xu, “Web services composition : A decade ’ s overview,” *Inf. Sci.*, vol. 280, pp. 218–238, 2014.

[8] M. Klusch, B. Fries, and K. Sycara, “OWLS-MX : A Hybrid Semantic Web Service Matchmaker for OWL-S Services,” *Web Semant. Sci. Serv. Agents World Wide Web*, vol. 7, no. 2, pp. 121–133, 2009.

[9] A. AlSedrani and A. Touir, “Web Service Composition Processes: a Comparative Study,” *Int. J. Web Serv. Comput.*, vol. 7, no. 1, pp. 1–21, 2016.

[10] M. Moghaddam and J. G. Davis, “Service Selection in Web Service Composition: A Comparative Review of Existing Approaches,” in *Web Services Foundations*, First., A. Bouguettaya, Q. Z. Sheng, and F. Daniel, Eds. New York: Springer, 2014, pp. 321–346.

[11] A. KIM, M. KANG, C. MEADOWS, E. IOUP, and J. SAMPLE, “A Framework for Automatic Web Service Composition,” Washington, USA, 2009.

[12] J. Sangers, F. Frasincar, F. Hogenboom, and V. Chepegin, “Semantic Web Service Discovery Using Natural Language Processing Techniques,” *Expert Syst. Appl.*, vol. 31, pp. 1–27, 2013.

[13] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, and B. Parsia, “OWL-S: Semantic Markup for Web Services,” *W3C Member Submission*, 2004. .

[14] G. Friedrich, M. Fugini, E. Mussi, B. Pernici, and G. Tagni, “Exception Handling for Repair in Service-Based Processes,” in *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 2010, vol. 36, no. 2, pp. 198–216.

[15] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, “QoS-Aware Replanning of Composite Web Services,” in *IEEE International Conference on Web Services (ICWS’05)*, 2005, pp. 121–129.

[16] T. Simpson and T. Dao, “WordNet-based semantic similarity measurement,” *The Code Project Open License (CPOL)*, 2010. [Online]. Available: http://www.codeproject.com/Articles. [Accessed: 01-Jan-2016].