A Web Service Composition Framework Based on Centrality and Community Structure

Sophea Chhun, Kanokwan Malang, Chantal Cherifi, Néjib Moalla, Yacine Ouzrout

A Web Service Composition Framework based on Centrality and Community Structure

Sophea Chhun
Department of Information and Communication Engineering
Institute of Technology of Cambodia
Phnom Penh, Cambodia
sophea.chhun@itc.edu.kh

Kanokwan Malang, Chantal Cherifi, Néjib Moalla, Yacine Ouzrout
IUT Lumière – DISP Laboratory
University of Lumière Lyon2
Lyon, France
{firstname.lastname}@univ-lyon2.fr

***Abstract*— Reusing existing services for developing business process applications helps reducing the development time and cost. Besides, existing atomic services may not be able to answer users' requirements. Therefore, as manual composition takes time and is error prone, automatic service composition methods are needed. Their objective is to find the best composite service with minimum number of services and the best performance in term of Quality of Service (QoS). In this paper, a complex network based service composition framework is proposed. It is based on a network representation of the Web service space, making it possible to exploit efficiently various centrality properties and its community structure. It applies the A* search algorithm by considering different network centralities (degree, betweenness, closeness) together with various static and dynamic QoS attributes (execution time, availability and number of calls, changing values of QoS over different timestamps) in its heuristic. Furthermore, the network community structure is integrated in the process, in order to reduce the execution time and increase the accuracy of composition.**

Keywords*— Node centrality; Community structure; Web service composition; Complex networks; QoS

I. INTRODUCTION

The W3C defines a Web service as "a software application that help enterprises to integrate applications as a form of middleware based on XML artifacts". It provides a platform and language independent solution for software development. Web services can be used to implement business process applications. The evolution of existing business process applications or the development of new ones can require more complex Web services. However, developing new Web services is time consuming and costly. The solution is to reuse existing available Web services. Besides, if existing atomic services cannot respond to users' requests, service composition methods can be applied to generate composite services from existing atomic ones.

The service composition process is referred to a concept of integrating individual Web services to conduct complex business transactions based on functionality and performance constraints [1]. Due to the fast-growing number of Web services in real-world systems, the Web service composition process becomes a tremendous task. Several researchers have dealt with composition of Web services in recent years. Different service composition models have been proposed such as Petri Net, Logical Programming, Markov Process, Chaining algorithm, AI planning, Finite state automation, Workflow and Graphs [2]–[7]. The majority of those techniques suffer from scalability problem.

The graph model is gaining popularity in the community. It allows overcoming the problem of scalability faced by other models. Web services are a complex system when considering their interaction relationships for the composition process. This real-world system made of numerous interacting elements can be apprehended from the complex network perspective. Indeed, some recent works address the Web service composition problem as a graph search problem, by taking advantages of network properties.

Our work is in this line. A complex network based Web service composition framework is proposed. The Web service network is built using semantic matching between input and output parameters. The synonym problems are solved by the WordNet database. Note that the network is built offline to disconnect this time consuming task from the composition search task. The main contributions of this work are:

1. *The reduction of the search space by targeting the most suited community in the network*. Classically, in the area of Web services, the term "community" designates a set of Web services or operations with similar functionalities, and consequently being substitutable. In this work, we adopt the complex network definition of a community. It is based on the structure of the network itself. The community structure is a macroscopic property that reveals the native organization of individuals in complex systems [8]–[10]. Web services belonging to a same community preferentially interact; they are more likely to participate to the same compositions.

2. *The definition of a heuristic for A* that combines Web services non-functional and functional features (centrality, QOS, shortest path).* The A* search algorithm finds the best path from a source node to a goal node in the network in order to satisfy a user request. It integrates both local and global optimization when selecting the best service composition path. Node centralities [11] quantify the importance of a node within the network; the degree, the betweenness and the closeness are indicators of nodes popularity. Popular Web services are more likely to participate to numerous compositions or to provide a way to reach the goal more quickly. QoS attributes and the variation of their value over timestamps are the guarantee of composition performances. The matching will help to retrieve more accurate compositions. The shortest path helps to minimize the number of services in a composition path

The rest of the paper is organized as follows. Section I provides background elements. Section III presents the related works. Section IV describes the proposed framework for Web service composition. Section V deals with concluding remarks and future works.

II. BACKGROUND

A. Complex Network Properties

The investigation of numerous real-world systems, characterized by a decentralized and apparently unplanned evolution, showed that, irrespective of their origin, they share a common network structure. They typically exhibit the "small-world" property, i.e. most nodes are not neighbors of one another, but they can be reached from every other by a small number of links. Another salient feature is the distribution of the number of links. Many real-world networks are highly inhomogeneous with a few highly connected nodes and a large majority of nodes with low degree. Such networks with a power-law distribution of the degrees are called "scale-free". Clustering, also known as transitivity, quantifies how well connected are the neighbors of a node. It is a typical property of friendship networks, where two individuals with a common friend are likely to be friends. Assortativity expresses how nodes tend to associate together. Centrality reveals the importance of a node within a network. Centrality measures range from geometric measures (e.g. degree, closeness), path-based measures (e.g. betweenness) and spectral measures (e.g. eigenvector). Among those centralities, the most popular measures are the degree, the betweenness, and the closeness. The node degree refers to the number of incoming or outgoing links of a node. The node betweenness is obtained by counting the number of geodesics going through it. The node closeness corresponds to the average number of hops from other nodes [32]. The community structure indicates the presence of fairly independent groups of nodes with a high density of links between nodes of the same group, and a comparatively low density of links between nodes of different groups. Many community detection algorithms have been proposed in order to discover the community structure [9]. They organize the network into a set of overlapping or non-overlapping communities. Real-world complex networks are generally divided into independent sub-networks called components. The size of the largest one is an important quantity. It is generally a measure of the effectiveness of the network at doing its job.

B. Web Service Network Models

In order to define Web services networks, we consider a Web service as a distributed application that exports a view of its functionalities in terms of input and output. Hence, a Web service consists of a set of operations and their input and output parameters. Although more sophisticated interaction network models can be defined such as bipartite graphs or even hypergraphs, dependency and interaction networks are the two main models of composition networks. A dependency network is defined as a directed graph in which nodes represent the set of parameters and links materialize the operations. In other words, a link is created between each of the input parameters of an operation and each of its output parameters. An interaction network expresses the interaction relationships between Web services or between their operations. In this paper, we focus on interaction networks of operations. An interaction network of operation is a directed graph where the nodes are the set of operations and a link indicates that the output parameters of an operation provide the necessary information to invoke another one. One can consider different levels of interactions depending on the degree of information provided by the invoking operation. The partial interaction corresponds to the case where the invoking operation provides only a portion of the input parameters to the invoked operation. Full interaction represents the case where all the parameters are provided. In other words, in a full interaction network, a link is drawn from an operation i towards another operation j if and only if for each input parameter in $Input_j$, a similar output parameter exists in $Output_i$. Partial interaction networks are interesting to deal with optional input values, while full interaction is more suited to the situations where all data inputs are necessary. A central task in extracting composition networks is to determine if two parameters are similar. Indeed, similarity is used to decide if a link has to be drawn between two nodes. This task depends on the nature of the Web services description, i.e. syntactic or semantic. For syntactic descriptions, the similarity is assessed on parameters name through a syntactic matching. For semantic descriptions, the similarity is performed on the ontological concept associated to the parameters.

C. Web Service Network Properties

In [33], the authors investigate the topological properties of various Web services networks based on the SAWSDL-

TC1[1] collection of Web services descriptions. Results of the experiments reveal that the giant component of various types of networks contains the great majority of the nodes and of the links. Indeed, the number of nodes ranges from 85% to 95%, and from 95% to 99% for the links. Additionally, these networks exhibit the small-world and the scale-free properties. The presence of important nodes, hubs and authorities, has also been highlighted. Those Web services or operations having a great number of relationships benefit from a bigger popularity than others and play as cornerstones. In [29], the authors report that Web services networks exhibit a community structure. In this work, they compare the outputs of seven non-overlapping community detection algorithms (EdgeBetweenness, Louvain, Springlass, Eigenvector, Walktrap, Infomap and Labelpropagation). All of them exhibit a community structure, according to their modularity values. Recall that, networks with high modularity have dense connections between the nodes within communities, but sparse connections between nodes in different communities. Results show that Louvain, Spinglass and Infomap provide the highest value of modularity comparing to others. Additionally, they compared how the community structure correlates using clustering based measures. It appears that Louvain, Spinglass and Infomap are more consensual in terms of content, especially in the semantic operation network. In addition, the results are also very similar for the syntactic service operation network.

III. RELATED WORKS

Though not related to complex networks, the following three related works give insights on some important points for our concern. First, the definition of Web services graph is addressed. Second, useful information about graph search algorithms is provided. Third, the different types of the composition path (sequential or parallel) are treated. And last, the notion of QoS appears. Note that in [25] and [26] the authors also deal with QoS.

Talantikite et al. 2009 [12] propose an inter-connected service composition graph to support service discovery and composition processes. The graph is built offline, before any user request. The matching between parameters is computed from the similarity measure between ontological concepts. The algorithm generates four kinds of composition: (i) Simple: as an atomic service; (ii) Serial composition; (iii) Independent parallel composition and (iv) Dependent parallel composition. The algorithm can detect several alternative compositions. They can be ranked and selected according to three quality criteria: maximum cumulative similarity measure, minimum execution time and minimum space memory.

Shang et al., 2013 [7] propose an automatic service composition solution based on a bipartite directed graph. It contains two types of node: data node and service node. The service node represents Web services. It contains QoS information; the authors consider price and time. A service node links two data nodes. Data nodes represent input and output parameters. Four algorithms are proposed to select a composite service: Breadth-First Search (BFS), Depth-First Search (DFS), Price-based BFS (PBFS) and Time-based BFS (TBFS). From experiments, the authors conclude that BFS outperforms the others, while the worst one is DFS.

Yang et al., 2010 [13] find the optional service composition path by using a directed acyclic graph. The graph is defined as *G (V, A, QoS)*. *V* represents the set of candidate Web services, including the starting point Web service and the target Web service. *A* defines edges that link two Web services. Cost and execution time of Web services are considered as *QoS* parameters. In the graph, Web services that provide the same functionalities are grouped. Ant colony and the genetic algorithms are applied to find the best service composition path.

Some researchers focus on using a heuristic graph search algorithm as A*. A* algorithm leverages a heuristic value to guide the search process. A* (1) is basically represented by the combination of a cost function and a heuristic function.

$$f(n) = g(n) + h(n) \quad (1)$$

Where

g(n) is the past path-cost function calculated from the starting node to the current node *n*.

h(n) is the admissible heuristic estimate cost from the node *n* to the destination node [16]. It is known as the future path-cost function. Therefore, f(n) is the total cost of a service composition path, which is not discovered before the termination of the algorithm.

Rodriguez-Mier et al., 2012 [17] present an automatic Web service composition algorithm that uses a dependency graph. The algorithm generates composite services that can be executed in a sequence way and in a parallel way. The graph is built at the request time, and it is divided in a set of layers. Each layer contains all services that can be executed with the outputs of the previous one. Once the graph is created, the A* algorithm is applied in backward direction. The heuristic function considers only the length of the service composition path. Two optimization techniques are used: (i) removing the unused services in each layer and (ii) combined equivalent services which have the same f(n) value when the neighbors are being generated. Due to online graph building, the service discovery task can take a long time if the composite service network is complex.

Some researchers integrate complex network properties in the service composition process. Bansal et al., 2010 [18], propose a trust-based technique for finding the composite services. The Web service composition network is built as a directed acyclic graph. The construction of the graph is based on functional and non-functional attributes. Non-functional attributes come from Service Level Agreement (SLA) measures. The trust rating is defined in the SLA

[1] http://projects.semwebcentral.org/projects/sawsdl-tc/

document and calculated based on the degree centrality of each service.

The work from Xu et al., 2014 [11], presents a service selection method based on the functionality and on the credit of a service. The credit of a service is related to its popularity, influence and authority. Those aspects are evaluated by degree and closeness measures that help to quantify the service activities. The service with the higher credit score is the more active, the more influential service, and has the higher ability to reach the related shorter paths in a network. The relationship between services is calculated based on eigenvector and betweenness centralities. Note that this work is mainly related to selection rather than to composition.

Rostami et al., 2014 [20] propose a novel technique using clustering and ant colony algorithm for finding the optimal semantic Web service composition. The semantic network is created at design time. The authors use clustering for categorizing Web services. The clusters gather functionally similar Web services. The clustering is introduced in order to increase accuracy and to reduce response time of the service composition process. The cluster that has the highest similarity score with a client request is selected for performing composite services.

TABLE I. COMPARISON OF NETWORK BASED WEB SERVICE COMPOSITION PROPOSITIONS

Papers	Network centrality				QoS	Shortest Path	Clustering	Heuristic algorithm	Sequence (S)/ Parallel (P)
	Degree	*Closeness*	*Betweenness*	*Eigen vector*					
Hashemian and Mavaddat, 2005 [14]						X			S
Arpinar et al., 2005 [15]						X			S
Oh et al., 2009 [22]								X	S
Bansal et al., 2010 [18]	X				X	X			S
Rodriguez-Mier et al., 2011 [17]						X	X	X	S&P
AlShawi et al., 2012 [24]								X	S
Xu et al., 2014 [11]	X	X	X	X		X			S
Guluru and Niyogi, 2014 [27]					X	X			S&P
Rostami et al., 2014 [20]							X		

TABLE I presents the various criteria used in order to select the best service composition path in related research studies. The "Network centrality" column indicates if some centrality is used to guide the search process. The "QoS" column indicates if some QoS attributes are taken into account to choose some Web services. The "Shortest path" column indicates if this notion is integrated in the search algorithm. The "Clustering" column refers to whether or not the similar Web services nodes are categorized in order to reduce the search space. The "Heuristic algorithm" column is for the use or not of A*. The "Sequence/Parallel" column identifies whether the proposed algorithm can generate sequential composite services only or parallel composite services as well. We observe that complex network properties, i.e. network centralities like degree, closeness, betweenness and eigenvector, for Web service composition are still underused. The categorization of Web services is not well spread as well. Besides, the majority of the propositions take into account the QoS aspect and few of them propose parallel solutions for compositions. These results suggest to take into account network centralities as well as QoS criteria, and to calculate the shortest paths within a multi-criteria heuristic. The combination of criteria of different nature should improve the composition accuracy by avoiding for example to get the shortest path solution with poor service performances. One must also consider the changing of the QoS values over timestamps, which is more close to the real-world situation. Furthermore, due to the large amount of available Web services, and consequently to the size of the networks, there is a need to target the semantically appropriate Web services, and to reduce the search space [28], [29]. In other words, one must work in an environment in which Web services or their operations are organized in order to target the scalability issue.

IV. NETWORK-BASED COMPOSITION FRAMEWORK

A. Overview

Fig. 1 presents the proposed service composition framework with its different components. Web service descriptions are published in the Universal Description, Discovery and Integration (UDDI) registry. So far, UDDI does not allow storing QoS attributes of Web services. In order to fill this gap, WSOnto [31], [30] is an ontology designed to store both functional and non-functional properties of Web services. In the context of knowledge sharing and reuse, an ontology is a description of concepts and their relationships for the automatic processing of the information. WSOnto is populated upstream with functional attributes coming from UDDI registry. Non-functional QoS attributes are calculated from the data tracked during service execution. Service Composition Network, SCN, is the interaction network of operations. SCN is built offline from Web services that are stored in WSOnto. Giant SCN is the largest component of SCN. Accordingly to the studies presented previously on the size of the largest component, small components and isolated nodes are discarded from the SCN, and the largest component is kept. Community SCN is

the giant component on which the community structure has been identified using an appropriate community detection algorithm. In this work, we focus on non-overlapping community detection. Accordingly to the studies presented previously, Louvain, Spinglass and Infomap can be used. Then, the network labeling process associates a name and a list of keywords to each community.

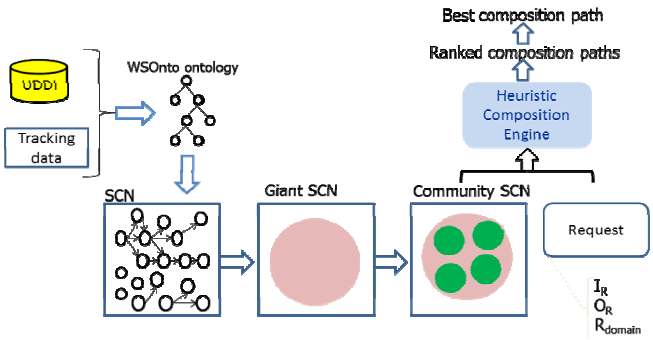

Fig. 1. Service composition framework

The Heuristic Composition Engine has several tasks. Its first task is to determine the best suited community for a given request. It takes the Community SCN and the request as inputs. The user's request is expressed by $R=\{I_R, O_R, R_{domain}\}$ where I_R are the input parameters, O_R are the output parameters, and R_{domain} is a list of keywords specifying the context of the request. Input and output parameters are expressed under the form of strings. Its second task is to run the composition algorithm on the satisfying community.

B. SCN Definition

SCN is defined by $G(V, E)$ where V represents the set of service operations and E represents the set of links between service operations. A service operation node $SO \in V$ contains service-id, operation's name, input and out parameters names, QoS values, QoS variation values and centrality values. Availability, execution time, and number of calls of a service operation are the three considered QoS attributes. The QoS variation represents the average change values of QoS over N number of times, based on its historical data. A link $e \in E$ represents the interaction between SO_i and SO_j. A source service operation SO_i connects to a goal service operation SO_j in the two cases defined hereafter:

- $SO_i^O = SO_j^I$: Outputs of SO_i are identical to inputs of SO_j in terms of number of parameters and matched string names.
- $SO_j^O \subset SO_i^I$: SO_i provides more parameters than what is requested by SO_j.

The comparison between parameters name is performed with the support of WordNet. The matching between two strings is done in two steps as follows:

- **Step1** Exact matching: It performs exact matching between two strings. It returns true if the two strings are identical and false otherwise.
- **Step2** Synonym matching: If the above step returns false, the comparison process continues by using WordNet. The synonym matching retrieves synonym words of a string name from WordNet, and compares them with another string name. If at least one match is found, the matching is considered successful.

C. Network Community Labeling

At its very first step, the Heuristic Composition Engine needs to find a matching between a user request and a community. To that end, the labeling process associates a name and a list of keywords to each community. A community is named with the keyword that has the maximum number of occurrences in the calculated network community. Keywords are extracted from service's name, service's description and service operation's name. In a keyword list, a keyword is identified by the keyword itself and the number of its occurrences in the community. If we assign only a name to a community without keeping the other keywords, the matcher might not be able to find a match community for a user. Note that it is possible to have several communities that match a user's request. In this case, the community with the maximum matching score, i.e. string similarity matching score, is considered.

D. Heuristic Composition Engine

In order to determine the best satisfying community regarding a given request, the Heuristic Composition Engine compares the context of the user with the keywords of each community. The community with the highest matched score is selected. The Heuristic Composition Engine is also responsible of finding the best composite service. To that end, its second task is to run the networked-based service composition algorithm on the satisfying community. It performs a forward chaining algorithm by applying the A* search algorithm. From the user's input parameters, I_R, the service operations that have the same or less inputs comparing to I_R, $SO_{Input} \leq I_R$, are determined. They will become the service sources. For each service source, the best composition path is calculated. The Heuristic Composition Engine selects the best composition path out of the candidate service composition paths, based on the maximum value of the f(n) function calculated by the A* search algorithm.

Fig. 2 introduces a pseudo code for building the satisfying Service Composition Network, satisfyingSCN. The idea is to build a network that contains all sub-networks, i.e. all possible composition paths that satisfy all determined source nodes. The building is done by using the breadth first search method. Line 8 finds the community that satisfies a user's request. After finding the satisfying community, the source nodes are determined in line 10. For each source node, the proposed algorithm finds a new sub-network that can derive from the source node to reach the goal node. The goal nodes are service operations that have

output parameters as the requested outputs or more parameters, line 12 to line 31. A list Q is created as a buffer to store nodes to treat, following the concept of BFS algorithm. The list Q is first empty at line 14 and it is initialized with a source node at line 15. The loop to construct child nodes of the finding sub network is done until the list Q is empty (line 16 to 28). If the sub network that reaches from a source node to goal nodes is not empty (line 29), add it into the satisfyingSCN. The algorithm returns the satisfyingSCN as output (line 32).

```
2  Input: SCN, R_input, R_output, R_domain
3  Output: SatisfyingSCN
4  // build service composite network based on the breadth first search algorithm
5  // that validate a user's request
6
7  // find the satisfy community
8  communityName <- determineCommunity(giantGraph, R_domain);
9  // find the source nodes
10 sources <- getSources(SCN, communityName, R_input);
11 // find composition subnetwork for each service source
12 for each source belong to sources do
13     H <- newNetwork(); // empty network
14     Q <- empty;
15     Q.push(source);
16     while Q != empty do
17         v <- Q.pop();
18         for all u belong to G.adjacenVector(V) do
19             if u != R_output
20                 if G.adjacentVentor(u)= empty then
21                     H <- removeNode(u);
22                 else
23                     Q.push();
24             else // u=R_output
25                 H <- addNode(v);
26                 H <- addChildNode(v, u); // v:parent, u:child
27         end for;
28     end while;
29     if H != empty then
30         satisfyingSCN <- addNetwork(H);
31 end for;
32 return satisfyingSCN;
```

Fig. 2. Pseudo code for building Satisfying Service Composition Network (SCN).

After the satisfying service composition network (satisfyingSCN) is built, composition paths can be defined. Fig. 3 presents the pseudo code of finding all possible composite paths. All source nodes are defined in line 6. For each source node, possible composition paths are determined (line 9 to line 18). Finally, line 21 chooses the best composition path based on the value of f(n) function of each path.

```
1  function: bestPath <- findBestPath(SCN, R_input, R_output);
2  Input: SCN, R_input, R_output
3  Output: bestPath
4
5  // find all services that have inputs the same with requested inputs
6  sources <- findSource(SCN, R_input);
7  if sources != emply then
8      // find the best path for each found source
9      for each s ϵ sources do
10         // walk in topological order
11         // build new network that satisfied to a request
12         newSCN <- buildNetwork(SCN, s);
13         // find the best path in each graph of the SCN
14         for each G ϵ newSCN do
15             result <- AStarSearch(newSCN, source);
16             listResult <- addPath(result);
17         end for;
18     end for;
19 end if;
20 // chose the best path besed on f(n) value of A*
21 path <- choseBestPath(listResult);
```

Fig. 3. Pseudo code to find all possible composite paths.

Fig.4 introduces the pseudo code of the A* search algorithm that is used to find the best composition path for a source node. The idea is to pass through the network, and at each step, to reach the goal node, choose the node with maximum value of f(n). At the initial state, the path is empty (line 5). From the current node which is the source node, the proposed algorithm finds the best next node (line 10 to 15), and assign it as current node. If the current node is the goal node (line 16), the best path is found (line 17). The f(n) value of the A* search algorithm is calculated at line 11. If the current node is not the goal node (line 18), the algorithm adds the current node to the graph. The algorithm returns the path with the f(n) value.

```
1  Function: resultpath <- AStarSearch (graph, source, R_output)
2  Input: graph G, source, R_output
3  Output: bestpath, fAstar
4
5  path <- empty;
6  currentNode <- source;
7  while bestpath not found do
8      hmax <- 0;
9      // find heuristic value of all candidate nodes
10     for all u ϵ G.adjacentVector(currentNode) do
11         hAstar <- heuristicValue(u);
12         if hAstar > hmax then
13             hmax <- hAstar;
14             currentNode <- u;
15     end for;
16     if currentNode = R_output
17         bestpath <- found;
18     else
19         path <- addPath(currentNode);
20     fAstar <- getgAstar(u) + hmax;
21 end while;
22 return path, fAstar;
```

Fig. 4. Pseudo code to find the best path by using the A* search algorithm.

The f(n) function of the A* search algorithm is defined in equation (2).

$$h(n)=V_{Centrality}+V_{qos}+V_{qosChange}+1/shortestPath \qquad (2)$$

Where

- $V_{Centrality}$ is the centrality value of a node. This centrality can be tuned by considering different centralities like degree, betweenness or closeness.
- V_{qos} defines a QoS value of a service operation that is calculated by using equation (3).
- $V_{qoschange}$ defines the QoS change value of service operations that is the average change value over different time periods. It is calculated by using equation (4).
- *shortestPath* defines the shortest path length from a node to the goal nodes.

QoS values define the performance of service operations. QoS attributes are divided into two categories. The first category concerns attributes that need to be maximized, while the second one is made of the attributes that need to be minimized. For example, availability and

number of calls have to be maximized, while execution time needs to be minimized. The overall QoS value is calculated according to equation (3).

$$QoS = w1*exec + w2*avail + w3*nbcalls \quad (3)$$

Where exec, avail and nbcalls respectively stand for execution time, availability and number of calls. wx are the weights provided by the user for each attribute.

QoS values of a service operation can change over timestamps. This dynamic behavior should be considered. To do so, QoSchange is calculated to get the variation of QoS over n numbers of period of times as follow:

1. First, calculate QoS at different period of times (t_0, t_1, t_2,…, t_n) using equation (3). The gap between two time slots t_i and t_{i+1} can be fixed for a period of one week or one month. Actually, a QoS value at the time t_i is an average QoS value during that time slot already.
2. After that, calculate the QoS variation between two QoS values at time t_i and time $t_{j=i+1}$: $PC_{ij}^{qos} = \frac{QoS_{new}}{QoS_{old}} - 1.$
3. Finally, calculate the average QoS change value: $\text{QoSchange} = \frac{\sum_{i=1;j=i+1}^{n-1} PC_{ij}}{n}$ (4).

The shortest path length criterion provides the minimum number of service operations in a composition path. It helps reducing the execution time of a composite service, since fewer operations are considered.

The composition framework has been implemented within an application dedicated to automatically implement business processes [19]. The case study is done on the SAWSDL-TC3 collection of Web service descriptions. Preliminary results show that A* performs better in terms of composition length, when the heuristic runs with the betweenness centrality. From the Web service composition perspective, the betweenness of a node is related to the number of shortest composition paths going through this node. Nevertheless, these findings must be tempered by the fact that the size of the network is relatively small (around 1000 nodes). For larger scale networks, the calculation of the betweenness centrality can be quite time-consuming. It may be more effective to use the degree centrality. Indeed, its computation is purely local. Furthermore, the algorithms for the community detection task should be tailored to large sized-networks.

V. Conclusion And Future Work

In this paper, we propose a framework for Web service composition. This framework is based on Web services complex networks topological features. It takes advantage of two salient complex network properties, the so-called community structure and centralities. A network is built from the descriptions of Web services that are available in an ontology specifically designed to support functional and non-functional, i.e. QoS, attributes. The communities in the network are detected by community detection algorithms. They are labeled by a name and a set of keywords coming from the attributes of the Web services and their operations. For a given request, the most suited community is targeted. A breadth first search method builds a network with all the possible composition paths from this community. This network is explored to find all the possible composition paths. The A* algorithm find the best composition path for a source node by applying local and global optimization. Local optimization uses A* in the forward chaining to find the best composition path from a source node to goal nodes. By using A* on each service source, its f(n) value is determined for each candidate composition paths. Global optimization is about selecting the composition path with the maximum value of f(n) among all candidate composition paths. The heuristic function of A* considers different criteria: network centralities, QoS values, change values of QoS, shortest path to reach requested goals.

The main contributions of the proposed approach are to tackle the issue of scalability caused by the large number of Web services and to obtain the best composite service with the minimum number of services with good performances. However, the current solution presents a main limitation. It can generate only sequential execution composite services. Our future work is to provide a solution to create parallel execution services.

Acknowledgment

This project has been funded with support from the European Commission, EMA2-2010- 2359 Project and Erasmus Mundus FUSION program 2014-2015. This publication reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

References

[1] R.-K. Sheu, W.-T. Lo, C.-F. Lin, and S.-M. Yuan, “Design and Implementation of a Relaxable Web Service Composition System,” 2010, pp. 448–455.

[2] J. Peer, “A PDDL based tool for automatic web service composition,” in Principles and practice of semantic web reasoning, Springer, 2004, pp. 149–163.

[3] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau, “HTN planning for Web Service composition using SHOP2,” Web Semant. Sci. Serv. Agents World Wide Web, vol. 1, no. 4, pp. 377–396, Oct. 2004.

[4] S. McIlraith and T. C. Son, “Adapting golog for composition of semantic web services,” KR, vol. 2, pp. 482–493, 2002.

[5] D. V. McDermott, “Estimated-Regression Planning for Interactions with Web Services.,” in AIPS, 2002, vol. 2, pp. 204–211.

[6] P. Doshi, R. Goodwin, R. Akkiraju, and K. Verma, “Dynamic workflow composition using markov decision processes,” in Web Services, 2004. Proceedings. IEEE International Conference on, 2004, pp. 576–582.

[7] J. Shang, L. Liu, and C. Wu, “WSCN: Web Service Composition Based on Complex Networks,” 2013, pp. 208–213.

[8] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D. Hwang, “Complex networks: Structure and dynamics,” Phys. Rep., vol. 424, no. 4–5, pp. 175–308, Feb. 2006.

[9] S. Fortunato and A. Lancichinetti, “Community detection algorithms: a comparative analysis: invited presentation, extended abstract,” in Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools, 2009, p. 27.

[10] M. E. Newman, “Detecting community structure in networks,” Eur. Phys. J. B-Condens. Matter Complex Syst., vol. 38, no. 2, pp. 321–330, 2004.

[11] B. Xu, J. Ge, and T. Chung, “Measuring Credit of Web Service,” 2014, pp. 423–430.

[12] H. N. Talantikite, D. Aissani, and N. Boudjlida, “Semantic annotations for web services discovery and composition,” Comput. Stand. Interfaces, vol. 31, no. 6, pp. 1108–1117, Nov. 2009.

[13] Z. Yang, C. Shang, Q. Liu, and C. Zhao, “A dynamic web services composition algorithm based on the combination of ant colony algorithm and genetic algorithm,” J. Comput. Inf. Syst., vol. 6, no. 8, pp. 2617–2622, 2010.

[14] S. V. Hashemian and F. Mavaddat, “A graph-based approach to web services composition,” in Applications and the Internet, 2005. Proceedings. The 2005 Symposium on, 2005, pp. 183–189.

[15] I. B. Arpinar, R. Zhang, B. Aleman-Meza, and A. Maduko, “Ontology-driven web services composition platform,” Inf. Syst. E-Bus. Manag., vol. 3, no. 2, pp. 175–199, 2005.

[16] K. Rana and M. Zaveri, “A-star algorithm for energy efficient routing in wireless sensor network,” in Trends in Network and Communications, Springer, 2011, pp. 232–241.

[17] P. Rodriguez-Mier, M. Mucientes, J. C. Vidal, and M. Lama, “An Optimal and Complete Algorithm for Automatic Web Service Composition:,” Int. J. Web Serv. Res., vol. 9, no. 2, pp. 1–20, 32 2012.

[18] S. K. Bansal, A. Bansal, and M. B. Blake, “Trust-based dynamic web service composition using social network analysis,” in Business Applications of Social Network Analysis (BASNA), 2010 IEEE International Workshop on, 2010, pp. 1–8.

[19] S. Chhun, “Dynamic Web Service Orchestration Architecture for the Re-Engineering of Business Processes", PhD. Thesis, 2015..

[20] N. H. Rostami, E. Kheirkhah, and M. Jalali, “An Optimized Semantic Web Service Composition Method Based on Clustering and Ant Colony Algorithm,” ArXiv Prepr. ArXiv14022271, 2014.

[21] H. N. Talantikite, D. Aissani, and N. Boudjlida, “Semantic annotations for web services discovery and composition,” Comput. Stand. Interfaces, vol. 31, no. 6, pp. 1108–1117, Nov. 2009.

[22] S.-C. Oh, J.-Y. Lee, S.-H. Cheong, S.-M. Lim, M.-W. Kim, S.-S. Lee, J.-B. Park, S.-D. Noh, and M. M. Sohn, “WSPR*: Web-Service Planner Augmented with A* Algorithm,” 2009, pp. 515–518.

[23] P. Rodriguez-Mier, M. Mucientes, and M. Lama, “Automatic Web Service Composition with a Heuristic-Based Search Algorithm,” 2011, pp. 81–88.

[24] I. S. AlShawi, L. Yan, W. Pan, and B. Luo, “Lifetime Enhancement in Wireless Sensor Networks Using Fuzzy Approach and A-Star Algorithm,” IEEE Sens. J., vol. 12, no. 10, pp. 3010–3018, Oct. 2012.

[25] P. Sun, P. Zhang, W. Li, X. Guo, and J. Feng, “Sky-MCSP-R: An Efficient Graph-Based Web Service Composition Approach,” 2013, pp. 589–594.

[26] G. Liu, Y. Zhao, Z. Wang, and Y. Liu, “A Service Chain Discovery and Recommendation Scheme Using Complex Network Theory,” Math. Probl. Eng., vol. 2014, pp. 1–6, 2014.

[27] P. Guluru and R. Niyogi, “New approaches for service composition based on graph models,” in Contemporary Computing (IC3), 2014 Seventh International Conference on, 2014, pp. 507–512.

[28] L. Cui, “Mathematical theory of service composition and service networks,” The Pennsylvania State University, 2011.

[29] C. Cherifi and J.-F. Santucci, “Community structure in interaction web service networks,” Int. J. Web Based Communities, vol. 9, no. 3, pp. 392–410, 2013.

[30] S. Chhun, C. Cherifi, N. Moalla, and Y. Ouzrout, “Business process implementation using an ontology-driven Web service selection algorithm,” presented at the 5th Journées Francophones sur les Ontologies (JFO 2014), Hammamet Tunisia, 2014, pp. 115–126.

[31] S. Chhun, N. Moalla, and Y. Ouzrout, “QoS ontology for service selection and reuse,” J. Intell. Manuf., pp. 1–13, 2014.

[32] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D. Hwang, “Complex networks: Structure and dynamics,” Phys. Rep., vol. 424, no. 4–5, pp. 175–308, Feb. 2006.

[33] C. Cherifi and J.-F. Santucci, “On Topological Structure of Web Services Networks for Composition,” International Journal of Web Engineering and Technology, Vol. 8, No. 3, pp.291-321, 2013