2010 IEEE International Conference on Services Computing

QoS Analysis and Service Selection for Composite Services

Huiyuan Zheng
Department of Computing
Macquarie University
Sydney, Australia
hyzheng@comp.mq.edu.au

Jian Yang
Department of Computing
Macquarie University
Sydney, Australia
jian@comp.mq.edu.au

Weiliang Zhao
Department of Computing
Macquarie University
Sydney, Australia
wzhao@comp.mq.edu.au

***Abstract*—A composite service can be constructed with the arbitrary combination of sequential, parallel, loop, and conditional structures. In this paper, we propose a general solution to calculate the QoS for composite services with complex structures. We also show QoS-based service selection can be conducted based on the proposed QoS calculation method. An application example is given to show the effectiveness of the method.**

***Keywords*-QoS; Web service composition; composition pattern**

I. Introduction

The nature of services creates the opportunity for building composite services by combining existing elementary or complex services (referred to as *component services*) from different enterprises and in turn offering them as value added services. QoS analysis becomes increasingly challenging and important when complex and mission critical applications are built upon services with different QoSs [1]. Thus solid model and method to support for QoS predication in service composition become crucial and will lay a foundation for further analysis of complexity and reliability in developing service oriented distributed applications.

It is important to estimate the QoS of a composite service at design time based on the quality of component Web services to make sure that the composition can satisfy the expectations of end users [2,3]. A Web service may need to be replaced with functionally equivalent services at run time if it becomes unavailable or its performance degrades too much [4]. A comparison is therefore necessary by analyzing the QoSs of the composite service with different service combination options.

Selecting service components for the composite service and making sure that the QoS of the composite service fulfil the user's QoS requirements is an optimization problem. QoS calculation is of great importance in the QoS-based service selection. A good QoS calculation method should:

(1) calculate the QoS for the composite service as a whole as well as the QoSs for alternative execution paths. User requirements are usually specified as the mean, the maximum, or the minimum of the QoS value. For example, the user requirement could be: the cost of the composite service A should not be over 5000 dollars and the cost of each execution should not exceed 7000 dollars. The former is a constraint on the mean cost while the latter is a constraint on the cost of individual paths. But current QoS calculation approach either computes QoS for the composite service or QoS for the paths, but not for both.

(2) be able to deal with complex structures. A composite service can be constructed based on basic patterns, such as sequential pattern, parallel pattern, and etc., which is referred to as *Composition Pattern* in this paper. Therefore, the QoS of a composite service can be seen as the aggregation of the QoS of these composition patterns. Existing QoS calculation approaches only deal with simple basic patterns. They can not handle the situation when basic patterns are combined in an arbitrary way. For example, a composite service has a parallel pattern nested in a loop pattern or a loop pattern nested in a parallel pattern.

Therefore, we need a mechanism which can identify patterns from the composite service even if they are nested in each other and be able to get the *composition information* for each execution path, such as its QoS and execution probability.

In our previous work [5], a QoS calculation method is proposed that can calculate the QoSs of the paths in a service composition with path execution probabilities. However, it has no capability to deal with complex structures.

This paper proposes a general solution to calculate the QoS of a composite service which can be modeled as directed graph with composition patterns arbitrarily combined. The proposed QoS calculation method can be used in QoS-based service selection. An example is given to show that the service selection result based on the proposed method is more effective in comparison with existing methods.

The remainder of the paper is organized as follows: Section II discusses the existing QoS calculation approach and their limitations. In Section III, the modelling approach of composite services and the definitions for basic composition patterns are introduced. An algorithm is given in Section IV to identify and process the composition patterns and generate QoS and probability for each path based on the graph model for the service. In Section V, the calculation result of the proposed method is applied in QoS-based service selection and the effectiveness of the proposed method is shown through an application senario. Finally, Section VI concludes the work.

II. Related Work

We will first look into the existing QoS aggregation approaches, then discuss calculation methods used in QoS-based service selection approaches.

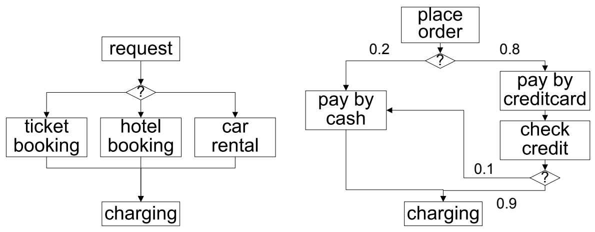
In work [6,7], a composite service is seen as being composed of basic composition patterns, such as sequential, parallel, loop, and conditional patterns. The QoS for a composite service can be treated as the aggregation of the QoSs of these patterns. These approaches have two major problems due to the way in which conditional patterns are handled:

(1) Only the mean QoS value is obtained and the probability information is lost. For example, if there are two services with the same mean QoS value, then the one with smaller deviation is better. This information cannot be obtained in [6,7];

(2) In general, conditional patterns can be represented in two ways: structured way and unstructured way.

`Structured way`: in which exclusive tasks are split from the same task and join together to another task. An example of structured condtional pattern can be seen in Figure 1(a). The QoS of it can be calculated as the probability weighted sum of the QoS of the service in each conditional branch [6,7].

`Unstructured way`: in which two or more paths can share the same tasks before they join together. For example, in Figure 1(b), task *PaybyCash* is shared by one branch splitting from task *PlaceOrder* and another branch splitting from task *CheckCredit*. All the branches finally join to task *Charging*.



(a) Structured (b) Unstructured

Figure 1. Example Service Processes of Conditional Patterns

[6,7] can only calculate QoS for structured conditional patterns. How to calculate the unstructured conditional patterns is still an open problem. When there are more of the unstructured conditional patterns, the QoS calculation for the composite service becomes very difficult. This problem can be solved by the proposed method in this paper.

There are also work on QoS-based service selection for Web service composition [2,3,8,9,10]. They can be classified into two categories depending on the QoS calculation method they use. They are:

(1) service selection methods [8,10] that calculate QoS based on the methods proposed in [6,7] whose limitations have been discussed earlier.

(2) service selection methods calculating the QoS of each path of the composite service [2,3,9]. But path probability information is not taken into account in the QoS calculation.

In [2,9], an optimal execution plan is selected for each path. Then these execution plans are merged to form an optimal execution plan for the composite service. The optimization is local to the execution paths instead of global to the composite service. Therefore, the result obtained by this method may not be the best solution.

In [3], it is required that the QoS of each path fulfills the global QoS requirements. The QoS for a composite service executing different paths varies. Imposing global constraints to every path makes the requirements too restrict to be satisfied.

III. Modeling of Service Process and Its Composition Patterns

In order to design an algorithm for QoS calculation, we need to formally define the basic composition patterns which a composite service is built upon.

A simple service graph is used to model a composite service. The vertices represent the component Web services and the edges denote the transitions with transition probabilities between services. The definition of the service graph is as follows:

Definition 1: Service Graph: Let S be the set of Web services, T be the set of transitions in a composite service, and P be the set of transition probabilities between two services linked by a transition. A Service Graph is $G = (V, A)$, where

- $V = S$ are the vertices of the graph;
- $A = T \subseteq V \times \triangledown \times V \times P$ are the arcs of the graph;
- $\triangledown = \left\{-, \|_{split}, \|_{join}\right\}$ are connection methods in the graph with
  - '$-$' denotes a sequential connection
  - '$\|_{split}$' denotes a concurrent split connection that will be followed by a synchronized merge connecion (the merge is triggered by the termination of all the concurrent running branches)
  - '$\|_{join}$' denotes a synchronized merge connection
- $\forall a_i \in A$, $a_i = (v_x \Phi v_y, p)$ where $\Phi \in \triangledown$ and $p \in P$, denoting that the arc from vertex v_x to v_y is a Φ (sequential, concurrent split, or synchronized merge connection) arc and the transition probability is p.

The representation of a service process by a directed graph is shown through the following example. Figure 2 is a composite service `OrderGoods` having nine tasks associated with execution probabilities.

According to Definition 1, the process in Figure 2 can be represented by a service graph: $G = (V, A)$ where $V = \{v_i | i \in [1, 9]\}$ and $A = \{(v_1 - v_2, 0.8), (v_1 - v_4, 0.2),$

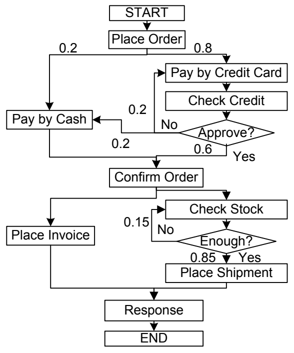


Figure 2. Service Process for `OrderGoods`

$(v_2-v_3,1), (v_3-v_2,0.2), (v_3-v_4,0.2), (v_3-v_5,0.6), (v_4-v_5,1), (v_5\|_{split}v_6,1), (v_5\|_{split}v_8,1), (v_6-v_6,0.15), (v_6-v_7,0.85), (v_7\|_{join}v_9,1), (v_8\|_{join}v_9,1)\}$. This service graph $G=(V,A)$ is depicted in Figure 3.

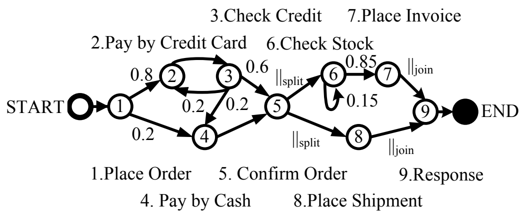


Figure 3. Service Graph for Example Service `OrderGoods`

In Figure 3, vertices 1 to 9 represent the nine Web services in the composite service of Figure 2. The arcs represent the transitions between services. The default connection of an arc is sequential connection and the default transition probability of an arc is 1. If the connection is not sequential or the transition probability is not 1, it will be explicitly specified in the graph. For example, the connection between vertices 5 and 6 is concurrent split, therefore, $\|_{split}$ is marked beside the arc from vertex 5 to 6. In the following sections, for the reason of clarity, we will all use the graph like Figure 3 to represent service graph $G=(V,A)$.

Workflow patterns have been discussed and defined in [11], among which we identify four bisic patterns of service composition: sequential, parallel, conditional, and structured loop patterns. The management of other patterns is similar to these basic ones.

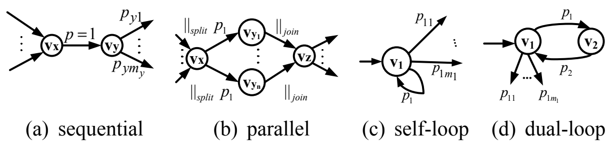


Figure 4. Basic Composition Patterns

Definition 2: Sequential Pattern (see Figure 4(a)): In $G=(V,A)$, $G'=(V',A')$ is a Sequential Pattern where $V'=\{v_x,v_y|indegree(v_y)=1\}$ and $A'=\{(v_x-v_y,1)\}$.

Three conditions have to be met when two adjacent vertices v_x and v_y composes a Sequential Pattern:

- the arc between v_x and v_y is a sequential connection;
- the transition probability from v_x to v_y equals to 1;
- there is only one incoming arc for vertex v_y.

Definition 3: Parallel Pattern (see Figure 4(b)): In $G=(V,A)$, $G'=(V',A')$ is a Parallel Pattern with n concurrently executed vertices where $V'=\{v_{y_i}|i\in[1,n]\}$ and $A'=\{(v_x\|_{split}v_{y_i},p_1),(v_{y_i}\|_{join}v_z,1)|i\in[1,n]\}$.

Only Parallel Pattern with synchronized merge mode is defined, i.e. the Web service will only be invoked when all branches are finished.

Definition 4: Structured Loop Pattern (Loop Pattern for short): In $G=(V,A)$,

- $G'=(V',A')$ is a Self-loop Pattern (see Figure 4(c)) if $V'=\{v_1\}$ and $A'=\{(v_1-v_1,p_1)\}$;
- $G'=(V',A')$ is a Dual-loop Pattern (see Figure 4(d)) if $V'=\{v_i|i\in[1,2]\}$ and $A'=\{(v_i-v_{i+1 \bmod n},p_i)|i\in[1,2]\}$.

In this paper, we assume that only structured loop, i.e. loop with only one entry and one exit point, exists in the directed graph. There are two basic types of Loop Patterns named as: `Self-loop` (the *repeat... until* loop in Figure 4(c)) and `Dual-loop` (the *while* loop in Figure 4(d)). There can be multiple vertices or even other patterns in a Loop Pattern. The algorithm proposed in this paper is able to transform the complex Loop Patterns first into the two basic ones and then into a single vertex. For space limit, we will only give the following example to show that any structured Loop Pattern can finally be transformed into either Self-loop or Dual-loop.

In Figure 3, there is a Loop Pattern having two vertices - vertex 2 and 3. It is not the same as any of the basic patterns in Figure 4. According to Definition 2, Vertex 2 and 3 compose a Sequential Pattern, and can be replaced by one vertex, say *X*. Then the original arc from vertex 3 to vertex 2, i.e. $(3-2,0.2)$, becomes an arc from *X* to *X*. Therefore, the structure composed of vertices 2 and 3 in Figure 3 can be transformed into a self-loop.

In this paper, instead of processing individual conditional structures, we will exploit the probability information of the arcs in conditional structures to compute the execution probability for every path of the service. We can get both the QoS for each path and the mean QoS value for the composite service.

Therefore, only three basic composition patterns Sequential, Parallel, and Loop are defined in service graph for the calculation of QoS. Here we will use the formulae developed in [5] to calculate the QoS for these patterns. The formulae are listed in Table I.

So far, the four basic composition patterns have been discussed. In order to record these patterns in their replacing vertices, the following definition is given:

Table I
QoS Calculation for Different Structures

	Sequential	Parallel	Self-loop	Dual-loop
Probability	probability of v_y	1	$p'_{1j} = \frac{p_{1j}}{1-p_1}(j \in [1, m_1])$	$p'_{1j} = \frac{p_{1j}}{1-p_1}(j \in [1, m_1])$
Cost	$C_{seq} = \sum_{i=1}^{n} C_i$	$C_{para} = \sum_{i=1}^{n} C_i$	$C_{s-loop} = \frac{C_1}{1 - p_1}$	$C_{d-loop} = \frac{C_1 + p_1 C_2}{1 - p_1 p_2}$
Time	$T_{seq} = \sum_{i=1}^{n} T_i$	$T_{para} = \max_{i \in [1,n]}(C_i)$	$T_{s-loop} = \frac{T_1}{1 - p_1}$	$T_{d-loop} = \frac{T_1 + p_1 T_2}{1 - p_1 p_2}$

Definition 5: $\psi = \{-, !, ||\}$ defines the relationships between vertices in graph $G = (V, A)$ with symbol "$-$" having the lowest priority level:

- the Sequential Pattern defined in *Definition 2* or two directly connected vertices in a conditional pattern can be represented by $v_x - v_y$;
- the Parallel Pattern defined in *Definition 3* can be represented by $v_{y_1} || v_{y_2} || \ldots || v_{y_n}$;
- the Self-loop Pattern defined in *Definition 4* can be represented by $v_1 ! v_1$; and the Dual-loop Pattern defined in *Definition 4* can be represented by $v_1 ! v_2$.

IV. Algorithm for Composite QoS Calculation

In this section, an algorithm is designed to process composite services modeled as service graphs with arbitrary combined patterns and calculate QoS for the service as well as QoS and probability for each possible execution path.

The input of the algorithm is the service graph for a composite service, as well as the QoSs of the component services. The output of the algorithm is the QoS and execution probability of each path of the service, and the QoS for the composite service. If the QoSs for component services are not given, the output of this algorithm contains the composition information for each path of the input graph which includes: (1)the probability of each path; (2) the vertices included in each path; (3) and the relationships between these vertices (see *Definition 5*). For example, the path containing vertices 1, 4, 5, 6, 7, 8 and 9 in Figure 3 will be output as $1 - 4 - 5 - (6!6 - 7)||8 - 9$. This output can then be used to generate the formulae to calculate the QoS for each path.

A. Algorithm Overview

In order to get every possible execution path of the composite service, a service graph needs to be transformed into a rooted tree in which the branches represent all the possible execution paths. During this transformation, each composition pattern is replaced by one single vertex which contains the following information: (1) the vertices and their relationships (see Definition 5) in the composition pattern so that the original paths of a composite service can still be tracked from the rooted tree even after the replacement and transformation processes take place; (2) the probabilities of the outgoing arcs of the composition pattern; and (3) the QoS of the composition pattern.

The formal way of exploring a graph is through depth first search (DFS) [12]. In this paper, we extend the recursive DFS algorithm using postorder traversal method so that the complex nested patterns can be handled. The processing for a composition pattern does not start immediately after this pattern is identified. Instead, it will be conducted after all the direct successors of the start point[1] of the pattern are fully explored.

Three information are needed to identify different composition patterns: (1) the status of a vertex, i.e. whether a vertex is *unvisited*, is in the middle of *visiting* (exploring its successors), or has been *visited*; (2) the connection way of edge, i.e. "SEQ" for sequential, "PARA_SPLIT" for concurrent split, and "PARA_JOIN" for synchronized merge; (3) and the transition probability of arc. Specifically: let vertex nxtV be vertex currV's direct successor, then nxtV belongs to a

- **Sequential Pattern**: if the edge between currV and nxtV is "SEQ", the transition probability from currV to nxtV is 1, and the indegree of nxtV is 1, then currV and nxtV compose a Sequential Pattern.
- **Parallel Pattern**: if the status of nxtV is unvisited and the edge from currV to nxtV is "PARA_SPLIT", then nxtV is a node in one branch of a Parallel Pattern; if the status of nxtV is visited and the edge from currV to nxtV is "PARA_JOIN", then nxtV is the node to which all the branches of a Parallel Pattern join.
- **Join Point for Two Paths**: if the status of nxtV is visited and the edge from currV to nxtV is "SEQ", then nxtV is the join point for two paths of a graph, i.e. two paths merge to the same path at nxtV.
- **Loop Pattern**: if the status of nxtV is visiting, then nxtV is the start point of a Loop Pattern and currV is its predecessor in the loop.

B. Algorithm

Algorithm 1 gives the solution to calculate QoS for composite services with arbitrarily combined patterns. Next, we will examine this algorithm in detail.

[1]The start point of a Sequential Pattern is the foremost vertex; of a Parallel Pattern is the vertex from which all the branches split; of a Loop Pattern is the entry point of the loop.

```
1  function DfsFrmGrph2Tr(Grph G)
2      headT = Root of Tree T;
3      headV = Head of Graph G;
4      IniGraph(headV); % initial graph G;
5      DfsVstGrph(headV); % G is turned into acyclic;
6      IniGraph(headV); % acyclic graph G;
7      DfsVstAcyclcGrph(headV, headT);
8  end
9  function IniGraph(GrphNd headV)
10     foreach vertex currV in Graph do
11         vertex_status[currV] = UNVISITED;
12     para_list[currV] = NULL;
13     loop_list[currV] = 0;
14 end
15 function DfsVstAcyclcGrph(GrphNd currV, TrNd fatherT)
16     sonT = Generate a tree node for currV;
17     add sonT to tree T as the child of fatherT;
18     foreach nxtV in adjacent[currV] do
19         if vertex_status[nxtV] == VISITED then
20             Copy the subtree rooted at nxtV to sonT;
21         else
22             DfsVstAcyclcGrph(nxtV, sonT);
23 end
24 function DfsVstGrph(GrphNd currV)
25     vertex_status[currV] = VISITING;
26     foreach nxtV in adjacent[currV] do
27         if vertex_status[nxtV] == UNVISITED then
28             if edge(currV, nxtV) == "PARA_SPLIT" then
29                 Add nxtV to para_list[currV];
30             DfsVstGrph(nxtV, sonT);
31         else if vertex_status[nxtV] == VISITING then
32             loop_list[nxtV] = 1;
33     Process the Sequential Pattern starting at currV if there is any;
34     if para_list[currV]!= NULL then
35         Process the Parallel Pattern stored in para_ list[currV];
36         Process the Sequential Pattern starting at currV if there is
           any;
37     if loop_list[currV] != 0 then
38         Process the Loop Pattern starting at currV;
39         Process the Sequential Pattern starting at currV if there is
           any;
40     node_status[currV] = VISITED;
41 end
```

Algorithm 1: Rooted Tree Generation

Function DfsFrmGrph2Tr is the main function that executes Algorithm 1. It transforms the service graph of a composite service into a tree which includes all the paths of the service graph and calculates the QoSs for both the service graph and its paths. There are two major functions in DfsFrmGrph2Tr which are function DfsVstGrph and DfsVstAcyclcGrph. Function DfsVstGrph removes all the Sequential, Parallel, and Loop Patterns from the input service graph and turns it into an acyclic graph with only conditional structures. Function DfsVstAcyclcGrph transforms this acyclic graph into a tree which contains all the possible execution paths of the composite service and calculates the QoS for each path and the composite service.

In function DfsVstGrph, the successors of a vertex are explored first when Parallel Pattern or Loop Pattern starting at this vertex is found (line 26 to 32). Then the processing of different patterns begins (line 33 to 39). During the processing of each pattern, the QoS of this pattern is calculated first according to the formulae in Table I. Then this pattern is replaced by a vertex with the calculated QoS. The replacing vertex records the IDs of the vertices it replaces and their composition patterns. The probability for the incoming arc of the vertex is the same as the pattern. For the outgoing arc/arcs of the vertex, the probability is/are calculated based on the formulae in Table I.

Function DfsVstAcyclcGrph transforms the acyclic graph into a tree. During the exploration of vertices (line 18) or the copy of subtrees (line 20), if one vertex is found to be the end point of the graph, then one path of the graph can be generated by back tracking the tree from the tree node of this vertex through to the root of the tree. After the exploration of the acyclic graph, all the paths are obtained. The vertices in each path compose a Sequential Pattern whose QoS can be calculated according to Table I. The probability of each path is the product of the probabilities of the arcs in the path. The QoS for the composite service is the probability weighted sum of the QoSs of each paths.

Here is an example to show the process of transforming a service graph with complex structures in Figure 3 into a tree with the paths information of the composite service included in the corresponding paths of the tree. Figure 5 and 6 show the transforming process. Color gray and black mean the status of a node being visiting and visited respectively. Unvisited node is not shown in the figure.

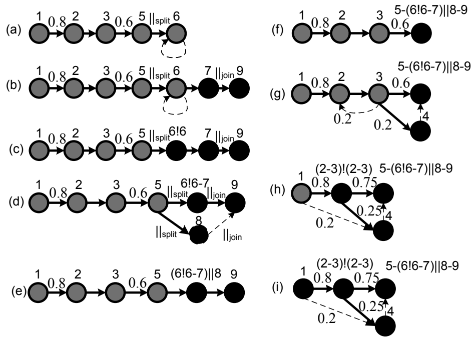


Figure 5. From Graph to Acyclic Graph - processing of Sequential, Loop, and Parallel Patterns

The algorithm keeps visiting vertex 1, 2, 3, 5, 6 orderly and their colors turn into gray (Figure 5(a)). Before the discovery of vertices 7 and 9, a Loop starting at vertex 6 is found (denoted by dashed line in Figure 5(a)). But as the direct successors of vertex 6 are not yet fully explored, nothing is done to the Loop. Followingly, vertex 9 and vertex 7 are explored (Figure 5(b)).

Then the loop at vertex 6 gets replaced by vertex 6!6 (Figure 5(c)). A Sequential Pattern is found started at vertex 6!6. Vertex 6!6 and vertex 7 are replaced by vertex 6!6 – 7.

The algorithm backtracks to vertex 5. After vertex 8 is fully explored (Figure 5(d)), the processing for the Parallel Pattern begins: vertex 6!6 – 7 and vertex 8 are replaced

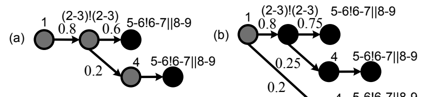


Figure 6. From Acyclic Graph to Tree - processing Conditional Patterns

by vertex (6!6 – 7)||8, and the incoming and the outgoing connections of vertex (6!6 – 7)||8 are changed to sequential connections (Figure 5(e)). Then vertex 5 – (6!6 – 7)||8 – 9 replaces the Sequential Pattern composed of vertex 5, vertex (6!6 – 7)||8, and vertex 9.

After the exploration of vertex 3 completes, a Loop Pattern is found starting at vertex 2 and the status of vertex 4 changes to black (see Figure 5(g)).

During the exploration of vertex 2, a Sequential Pattern composed of vertex 2 and vertex 3 is first replaced by vertex 2 – 3, then the self-loop of vertex 2 – 3 is replaced by (2 – 3)!(2 – 3) (see Figure 5(h)).

Finally, an acyclic graph is obtained which only contains Conditional Patterns (see Figure 5(i)).

Next, we will explain how the Conditional Patterns in the acyclic graph are handled.

During the transformation from acyclic graph to tree, a copy of vertex 5 – (6!6 – 7)||8 – 9 is added as the child of vertex 4 when the status of vertex 5 – (6!6 – 7)||8 – 9 is found to be *VISITED* (Figure 6(a)). Same thing happens when the status of vertex 4 is found to be *VISITED*. A tree rooted at vertex 1 is finally generated (see Figure 6(b)). It can be seen that there are three paths in the tree of the composite service. The composition information of the path with probability 0.2 is 4 – 5 – (6!6 – 7)||8 – 9 through which the vertices in the path and the connections between them can be obtained. It is the same for the other two paths.

V. Application

One of the applications of QoS calculation is in QoS-based service selection. In this section, we will illustrate (1) how our QoS calculation method can be used in service selection; (2) the effectiveness for it to be used in service selection by comparing the selection results with other existing QoS calculation methods. For simplicity, we will use an acyclic graph to represent a composite service (named as *Goal Service*) and integer programming to select services for this goal service.

A. A Running Application

Let us look at a service selection scenario.

Figure 7 gives the goal service that needs to select services to realize its tasks. For each task of the goal service, there are several functionally equivalent Web services. The cost and execution time of these Web services are listed in Table II. For example, for task 1, there are three available services: the cost and execution time of Web service 1 are 1500 and 20; of Web service 2 are 1000 and 60; and of Web service 3 are 500 and 50 respectively.

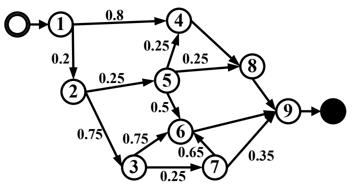


Figure 7. Goal Service

Table II
Available Web Services for the goal service in Figure 7

QoS / task	WS1		WS2		WS3	
	cost	time	cost	time	cost	time
1	1500	20	1000	60	500	50
2	600	60	100	100	-	-
3	800	40	-	-	-	-
4	900	20	1000	20	-	-
5	180	100	1500	20	-	-
6	800	40	2000	10	-	-
7	1200	20	2800	10	-	-
8	2000	10	4000	5	-	-
9	600	40	400	60	-	-

There are two requirements in relation to the goal service, which are:

`Requirements 1:` user's preferences over the QoS metrics cost and time are $w_{cost} = 0.7$ and $w_{time} = 0.3$ respectively. The cost of the service should not exceed 5000 and the execution time should not be over 120, i.e. $cost(C) \leq 5000$ and $time(T) \leq 150$.

`Requirements 2:` besides the requirements specified in `Constraint 1`, the cost of each execution of the composite service should not exceed 7500, and the execution time of each execution of the composite service should be less than 200.

Based on these user requirements, composition plans of the Web services in Table II will be generated to execute the goal service.

Next, we will show how the composition plans for the goal service are generated to fulfill the user requirements based on different QoS calculation methods. Then the QoSs of these composition plans will be compared.

B. Execution Path Generation

In this section, we shall use Algorithm 1 to transform the goal service in Figure 7 into a tree. The result is shown in Table III where the information of each path can be seen. For example, the execution probability of Path 7 is 0.8. The vertices in this path are vertex 1, 4, 8, and 9 and they are sequentially connected.

Based on the path information in Table III and the QoS calculation formulae in Table I, the formulae to calculate the QoS for each path can be formed.

The cost of each path is:

$$cost(path_i) = \sum_{j \in path_i} c_j \qquad (1)$$

Table III
Path List for The Goal Service in Figure 7

Path ID	Composition of Paths	Probability
1	1-2-5-4-8-9	0.0125
2	1-2-5-8-9	0.0125
3	1-2-5-6-9	0.025
4	1-2-3-6-9	0.1125
5	1-2-3-7-6-9	0.0244
6	1-2-3-7-9	0.0131
7	1-4-8-9	0.8

where j is the ID of the vertex in the path and c_j is the cost of vertex j. For example, the cost of Path 7 $cost(path_7)$ is $c_1 + c_4 + c_8 + c_9$.

The mean cost of the composite service is

$$mean(cost) = \sum_{i=1}^{7} p_i cost(path_i) = \sum_{i=1}^{7}(p_i \sum_{j \in path_i} c_j)$$

$$= c_1 + 0.2c_2 + \ldots + 0.825c_8 + c_9 = \sum_{k=1}^{9} M_k c_k \quad (2)$$

where M_k is a constant.

Let c_{km} be the cost of the m_{th} available Web service for vertex k and N_k be the number of available Web services for vertex k. We use $x_{km} = 1$ to represent the m_{th} Web service for vertex k is selected and $x_{km} = 0$ otherwise. There is $\sum_{m=1}^{N_k} x_{km} = 1$. Then the cost of each vertex c_k in Formula 2 can be represented as:

$$c_k = \sum_{m=1}^{N_k} x_{km} c_{km} \quad (3)$$

From Formulae 1, 2, and 3, there are:

$$cost(path_i) = \sum_{j \in path_i} \sum_{m=1}^{N_j} x_{jm} c_{jm} \quad (4)$$

$$mean(cost) = \sum_{k=1}^{9}(M_k(\sum_{m=1}^{N_k} x_{km} c_{km})) \quad (5)$$

For the same reason, let t_k be the execution time of vertex k and t_{km} be the execution time of the m_{th} available Web services for vertex k, there are:

$$t_k = \sum_{m=1}^{N_k} x_{km} t_{km} \quad (6)$$

$$time(path_i) = \sum_{j \in pathi} t_j = \sum_{j \in pathi} \sum_{m=1}^{N_j} x_{jm} t_{jm} \quad (7)$$

$$mean(time) = \sum_{k=1}^{9}(M_k(\sum_{m=1}^{N_k} x_{km} t_{km})) \quad (8)$$

C. Service Selection

In this part, service selection is conducted to generate composition plans for the goal service to fulfill the requirements.

Composition Plan 1 Satisfies `Requirements 1`

`Requirements 1` only sets requirements on the mean QoS values.

We will first conduct service selection using our method. Then the result is compared with the ones generated by other existing methods.

Using the proposed method in this paper, the formulae of calculating the mean QoS values are Formulae 5 and 8.

The objective function is:

$$Min(w_{cost} \times \frac{mean(cost) - MIN_{cost}}{MAX_{cost} - MIN_{cost}} + w_{time} \times \frac{mean(time) - MIN_{time}}{MAX_{time} - MIN_{time}}) \quad (9)$$

The constraints of the optimization problem can be expressed as:

$$\sum_{k=1}^{9}(M_k(\sum_{m=1}^{N_k} x_{km} c_{km})) \leq 5000 \quad (10)$$

$$\sum_{k=1}^{9}(M_k(\sum_{m=1}^{N_k} x_{km} t_{km})) \leq 150 \quad (11)$$

$$\sum_{m=1}^{N_k} x_{km} = 1 \quad (k = 1, 2, \ldots, 9) \quad (12)$$

The result for this optimization problem is composition plan 1 to execute the goal service which is shown in Table IV. Composition plan 1 lists the selected services for vertices 1 to 9 respectively.

Table IV
Composition Plans for User Requirements

vertex	①	②	③	④	⑤	⑥	⑦	⑧	⑨
comp.1	W3	W1	W1	W1	W2	W1	W1	W1	W1
comp.2	W3	W1	W1	W1	W2	W2	W2	W1	W1

In Table IV, the first row lists the IDs of vertices, and the second row lists the selected Web services for corresponding vertices based on the service information in Table II. For example, Web service 3 of vertex 1 is selected and its cost and execution time are 500 and 50 respectively.

After the composition plan of the goal service is selected, the QoSs of the composite service can be computed by Algorithm 1. The results are shown in Figure 8(a) and 8(b). It can be seen that the mean cost is 3970.75 and the execution time is 140.72 which all satisfy `Requirements 1`.

As for approaches that calculate the QoS of each path [2,3, 9], it is required that the QoS of each path fulfills the global QoS requirements. Imposing global constraints to every path makes the requirements too restrict to be satisfied. It can be seen through the following optimization process. Formulae 3 and 6 can represent the QoS calculation results for each path. For each generated path in Table III, the constraints can be set as follows:

$$cost(path_i) = \sum_{j \in path_i} \sum_{m=1}^{N_j} x_{jm} c_{jm} \leq 5000$$

$$time(path_i) = \sum_{j \in pathi} \sum_{m=1}^{N_j} x_{jm} t_{jm} \leq 150 \quad (13)$$

$$\sum_{m=1}^{N_j} x_{jm} = 1$$

There are some paths with no optimal solution. For example, the minimum execution time of Path 1 is 165 which does not fulfill the constraint in Formula 13. No execution plan can be generated by adopting these QoS calculation methods. Furthermore, the execution probability for Path 1 is only 0.0125 which means that the constraints are too strict when using this type of QoS calculation methods.

Composition Plan 2 Satisfies `Requirements 2`

`Requirments 2` also specifies requirements on the QoSs of paths, i.e. the maximum cost of each path is 7500 and the maximum execution time of each path is 200.

The optimization problem can be expressed as follows:

Objective function: the same as Formula 9.

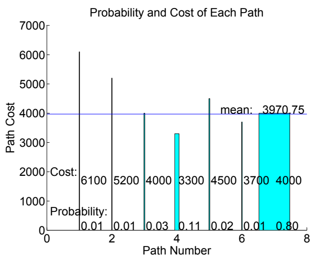
Constraints: besides Formulae 10, 11, and 12, there are:

$$cost(path_i) = \sum_{j \in path_i} \sum_{m=1}^{N_j} x_{jm} c_{jm} \leq 7500 \quad (14)$$

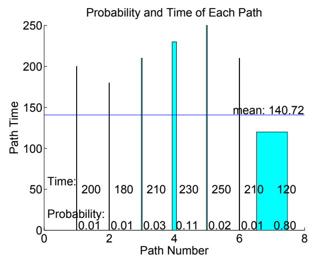
$$time(path_i) = \sum_{j \in pathi} \sum_{m=1}^{N_j} x_{jm} t_{jm} \leq 200 \quad (15)$$

The selection result of the above optimization problem is shown in the third row of Table IV.

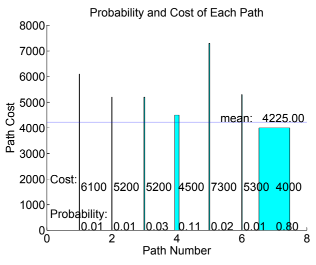
After the composition plan is settled, the QoSs of the composite service can be computed by Algorithm 1. The results are shown in Figure 8(c) and 8(d).



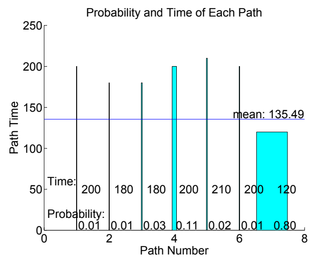
(a) Cost for Composition Plan 1



(b) Time for Composition Plan 1



(c) Cost for Composition Plan 2



(d) Time for Composition Plan 2

Figure 8. Service Selection Results

By comparing the two sets of QoSs of `Composition Plan 1` and `Composition Plan 2`, it can be seen that though the cost of `Composition Plan 2` is a bit higher than that of `Composition Plan 1`, the deviation of the execution time of `Composition Plan 2` is smaller. Furthermore, the cost of `Composition Plan 2` is still within the required limits.

Through the example in this section, it can be seen clearly that service selection can be more effective by using the proposed QoS calculation method.

VI. Conclusion

In this paper, we propose a QoS calculation approach which is able to calculate the QoS for individual execution paths of a composite service as well as the composite service even with the existence of complex composition structures. By using the proposed QoS calculation method in QoS-based service selection, users are able to explicitly specify their requirements on the mean, the maximum, and the minimum QoS value; the QoS value of the path with the largest execution probability; and even the deviation of the QoS. In the future, different service selection methods will be studied that can make full use of the obtained QoS information.

References

[1] “Qos for web services: Requirements and possible approaches,” W3C Working Group, Tech. Rep. 25, November 2003.

[2] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, “Qos-aware middleware for web services composition,” *IEEE Transactions on Software Engineering*, vol. 30, no. 5, pp. 311–327, May 2004.

[3] D. Ardagna and B. Pernici, “Adaptive service composition in flexible processes,” *IEEE Transactions on Software Engineering*, vol. 33, no. 6, pp. 369–384, June 2007.

[4] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, “Qos-aware replanning of composite web services,” in *ICWS ’05*, 2005, pp. 121–129.

[5] H. Zheng, W. Zhao, J. Yang, and A. Bouguettaya, “Qos analysis for web service composition,” in *SCC ’09*, 2009, pp. 235–242.

[6] J. Cardoso, J. Miller, A. Sheth, and J. Arnold, “Quality of service for workflows and web service processes,” *Journal of Web Semantics*, vol. 1, pp. 281–308, 2004.

[7] M. Jaeger, G. Rojec-Goldmann, and G. Muhl, “Qos aggregation for web service composition using workflow patterns,” *EDOC ’04*, pp. 149–159, Sept. 2004.

[8] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, “An approach for qos-aware service composition based on genetic algorithms,” in *GECCO ’05*, 2005, pp. 1069–1075.

[9] M. Alrifai and T. Risse, “Combining global optimization with local selection for efficient qos-aware service composition,” in *WWW ’09*, 2009, pp. 881–890.

[10] V. Cardellini, E. Casalicchio, V. Grassi, F. Lo Presti, and R. Mirandola, “Qos-driven runtime adaptation of service oriented architectures,” in *ESEC/SIGSOFT FSE ’09*.

[11] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros, “Workflow patterns,” *Distributed and Parallel Databases*, vol. 14, no. 1, pp. 5–51, 2003.

[12] M. A. Weiss, *Data structures and algorithm analysis in C*, 2nd ed. USA: Addison-Wesley Longman Publishing Co., Inc., 1997.