University of Wollongong
Research Online

Faculty of Engineering - Papers (Archive)

Faculty of Engineering and Information Sciences

2009

QoS analysis for web service composition

Huiyuan Zheng
Macquarie University

Weiliang Zhao
Macquarie University, wzhao@uow.edu.au

Jian Yang
Macquarie University

Athman Bouguettaya
CSIRO

http://ro.uow.edu.au/engpapers/5210

Publication Details

Zheng, H., Zhao, W., Yang, J. & Bouguettaya, A. (2009). QoS Analysis for Web Service Composition. IEEE International Conference on Service Computing 2009 (pp. 235-242). Australia: IEEE.

Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au

2009 IEEE International Conference on Services Computing

QoS Analysis for Web Service Composition

Huiyuan Zheng, Weiliang Zhao, Jian Yang
Department of Computing
Macquarie University
Sydney, Australia
{*hyzheng, wzhao, jian*}*@comp.mq.edu.au*

Athman Bouguettaya
ICT Centre
CSIRO
Canberra, Australia
Athman.Bouguettaya@csiro.au

***Abstract*—The quality of service (QoS) is a major concern in the design and management of Web service composition. Existing methods for QoS calculation either do not take the probability of path execution into consideration when QoSs are provided for different execution paths, or do not take different execution paths into consideration when a single integrated QoS is provided for the whole composition. In this paper, a comprehensive QoS analysis approach is proposed that calculates the QoS probability distribution by considering both the execution probability and execution conditions of each path in the service composition. Four types of basic composition patterns in the service composition are discussed: sequential, parallel, loop and conditional. In particular, a QoS solution is provided for all types of loop structured service composition.**

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

I. Introduction

Web service composition is emerging as a technique of choice for modelling and implementing business processes. With the fast adoption of Web services, one of the vexing challenges is analyzing and understanding the QoS of composite services[1][10].

It is important to estimate the QoS of a composite service at design time based on the quality of individual Web services to make sure that this composition can satisfy the expectations of end users. A Web service will be replaced at run time if it becomes unavailable or its performance degrades too much[5]. The impact of a replacing service to the performance of the composition should be evaluated before the replacement takes place. Normally, functionally equivalent services may exist with different QoSs. A comparison is therefore necessary by analyzing the QoS of the composite service with different services.

Based on the way of the results being represented, existing QoS calculation methods can be classified into two categories: (1) Reduction method with single QoS for the service composition[6][8]; (2) Direct aggregation method with multiple QoSs for the service composition[3][11].

For the reduction method, the process of a composite service is seen as an integration of different types of basic composition patterns. Though the calculation methods are different for these patterns, the basic idea about QoS calculation is the same, which recursively merges the tasks of a composite service till one single compound task is reached. The QoS for this single task is referred to as the QoS of the composite service.

The direct aggregation method is used in QoS driven Web service selection to calculate the QoS of each execution path of the composite service. The methods used in Web service selection make sure that the QoS of every path can satisfy the user's requirements.

Existing QoS calculation methods are not able to provide a complete picture of the QoS for a composite service. Their limitations will be illustrated through the following example.

Figure 1 is the process of a composite service with two paths. The execution probabilities of these two paths are $p_1 = 0.6$ and $p_2 = 0.4$ respectively. There is one task in each path. The cost and response time for task1 are C_1 and T_1, for task2 are C_2 and T_2. For each task, there are two functionally equivalent Web services to choose. The QoS (response time and cost) of these available Web services are shown in Table I. The user specified requirements on QoS are: the response time should be less than 50 and the cost should be no more than 25.

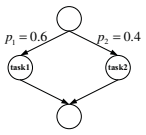


Figure 1. A Composite Service

Table I
QoS for Available Web Services

Task	Available WS	Response Time	Cost
Task1	WS1	10	30
	WS2	40	25
Task2	WS3	60	20
	WS4	100	10

First, we will show the limitations of using single values to represent the QoS of service composition. We adopt the method in [6] (see Formulae 1 and 2) to calculate the QoS for the conditional composition pattern shown in Figure 1.

$$Time\ of\ Service \quad T = p_1T_1 + p_2T_2 \tag{1}$$

$$Cost\ of\ Service \quad C = p_1C_1 + p_2C_2 \tag{2}$$

978-0-7695-3811-2/09 $26.00 © 2009 IEEE
DOI 10.1109/SCC.2009.28

In view of Formulae 1, 2, and Table I, there are four possible compositions:

1) `Composition 1`: WS1 and WS3 with $T = 30$ ($T = 0.6 \times 10 + 0.4 \times 60 = 30$) and $C = 26$ ($C = 0.6 \times 30 + 0.4 \times 20 = 26$);
2) `Composition 2`: WS1 and WS4 with $T = 46$ and $C = 22$;
3) `Composition 3`: WS2 and WS3 with $T = 48$ and $C = 23$;
4) `composition 4`: WS2 and WS4 with $T = 64$ and $C = 19$.

Two of them can meet the user specified requirements, which are `Composition 2 and Composition 3`. Since the response time and cost of `Composition 2` are both smaller than those of `Composition 3`, `Composition 2` is a better candidate for the composite service.

However, if the detailed QoS probability distribution information is provided, e.g. for `Composition 2`, the response time is 10 for 60% of the time and 100 for 40% of the time; for `Composition 3`, the response time is 40 for 60% of the time and 60 for 40% of the time, `Composition 3` is more likely to be selected. Simply because an extreme short response time as 10 is not needed and the long response time as 100 can not be tolerated.

From this example, it shows that the lack of information about execution probability and QoS for each path prevents the selection of the best execution plan in practice, and single values are not enough to represent the QoS of service compositions.

Next, we will show the limitations of the direct aggregation method. By adopting the Direct Aggregation Method, each path in Figure 1 should satisfy the requirements on QoS. WS1 with cost 30, WS3 with response time 60, and WS4 with response time 100 all exceed the required limits. The designed service composition can not be realized, because only WS2 can be selected for task1 and no service are suitable for task2.

The direct aggregation method overlooks the path execution probability of the composite service. There can be ups and downs in the QoS for a composite service to execute different paths. Therefore, it is only reasonable that QoS is based on statistic values. Adding constraints to every path without considering the execution probability makes the user specified requirements too difficult to be satisfied.

In this paper, a QoS calculation approach is proposed that does not only provide the QoSs of multiple paths for a service composition but also includes the execution probability of each path. Our main contributions in this paper are: (1) A service graph is defined to represent the process of a composite service; (2) QoS probability distribution is provided in the final result for the analysis of a composite service; (3) Four types of composition patterns are formally defined, and QoS and probability calculation methods for them are provided; (4) It is the first work to provide a solution to calculate the QoS for all types of loop patterns.

The remainder of the paper is organized as follows: Section II discusses four types of basic composition patterns. In the subsequent sections, the detailed process of QoS analysis will be introduced and a running example will be given. This paper ends with a discussion about the related work and the concluding remarks.

II. Building Blocks of a Composition

There are different composition patterns in the process of a composite service[8]. In this paper, we consider that a composite service is composed of four basic composition patterns: sequential, parallel, conditional, and loop (see Figure 3).

A. Preliminaries

In this paper, a composite service is represented by a directed graph which is referred to as Service Graph.

Definition 1. Service Graph: *Let S be the set of Web services and D be the set of dependencies within the process of a composite service, P be the set of transition probabilities from each service to its adjacent services within the composition. A Service Graph is $G = (V, A)$, where*

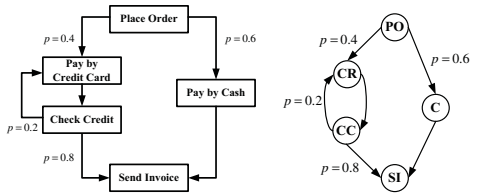
- *$V = S$ are the vertices of the graph;*
- *$A = D \subseteq V \times \triangledown \times V \times P$ are the arcs of the graph;*
- *$\triangledown = \{-, =\}$ are the two ways of connection in the graph with*
  - *'$-$' denoting sequential-connected way*
  - *'$=$' denoting concurrent-connected way*
- *$\forall a_i \in A$, $a_i = (v_x \Phi v_y, p)$ where $\Phi \in \triangledown$ and $p \in P$ with*
  - *$(v_x - v_y, p)$ denoting that the arc from vertex v_x to v_y is a sequential-connected arc and the transition probability is p for this arc*
  - *$(v_x = v_y, p)$ denoting that the arc from vertex v_x to v_y is a concurrent-connected arc and the transition probability is p for this arc.*

Definition 2. Indegree and Outdgree of a vertex: *for an arc $a = (v_x \Phi\ v_y, p)$, where $\Phi \in \triangledown$, v_x and v_y are the endpoints of arc a, v_y is the head endpoint of arc a, and v_x is the tail endpoint of arc a. For a vertex v, the number of head endpoints adjacent to v is called the indegree of v. The number of tail endpoints is its outdegree. The indegree is denoted as $deg^-(v)$ and the outdegree as $deg^+(v)$.*

For example, the process of a composite service in Figure 2(a) can be represented by a Service Graph $G = (V, A)$ in Figure 2(b), where $V = \{v_{PO}, v_{CR}, v_{CC}, v_C, v_{S1}\}$ and $A = \{(v_{PO} - v_{CR}, 0.4), (v_{PO} - v_C, 0.6), (v_{CR} - v_{CC}, 1), (v_{CC} - v_{CR}, 0.2), (v_{CC} - v_{S1}, 0.8), (v_C - v_{S1}, 1)\}$. The indegree and outdegree of vertex v_{CR} are: $deg^-(v_{CR}) = 2$ and $deg^+(v_{CR}) = 1$. In Figure 2, only p with a probability not equal to 1 is marked out.

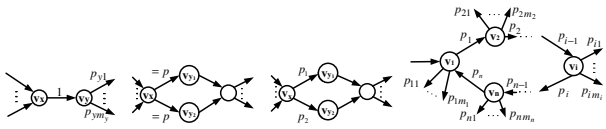
All the following discussion will be based on the Service Graph defined above.

Next, we will discuss the four types of composition patterns in Service Graph which will be used as building blocks of a composite service. Rather than providing a panacea for any QoS attribute, our approach only focuses on two representative QoS attributes: cost and availability.



(a) Service Process (b) Service Graph

Figure 2. Service Graph of a Composition



(a) Sequential (b) Parallel (c) Conditional (d) Loop

Figure 3. Building Blocks

B. Sequential Pattern

Definition 3. Sequential Vertices *(see Figure 3(a)): for G = (V, A), $\forall v_x, v_y \in V$, if $\exists(v_x - v_y, 1) \in A \wedge deg^+(v_x) = 1 \wedge deg^-(v_y) = 1$, then v_x and v_y are Sequential Vertices. Vertices v_x and v_y form a composition of Sequential Pattern.*

The transition probability of v_y is $P_y = \left\{p_{yi} \mid i \in [1, m_y]\right\}$. The cost of v_x and v_y are c_x and c_y respectively. The availability of them are a_x and a_y respectively. The transition probability (P'), cost (c'), and availability (a') for the composition of Sequential Pattern are:

$$P' = P_y = \left\{p_{y_i} \mid i \in [1, m_y]\right\} \quad (3)$$

$$c' = c_x + c_y \quad (4)$$

$$a' = a_x \times a_y \quad (5)$$

C. Parallel Pattern

Definition 4. Parallel Vertices *(see Figure 3(b)): for G = (V, A), $\forall v_x, v_{y_1}, v_{y_2} \in V$, if $\exists(v_x = v_{y_1}, p) \in A \wedge \exists(v_x = v_{y_2}, p) \in A$, then v_{y_1} and v_{y_2} are Parallel Vertices. Vertices v_{y_1} and v_{y_2} form a composition of Parallel Pattern.*

The cost of v_{y_1} and v_{y_2} are c_{y_1} and c_{y_1} respectively. The availability of them are a_{y_1} and a_{y_2} respectively. The transition probability (p'), cost (c'), and availability (a') for the composition of Parallel Pattern are:

$$p' = 1 \quad (6)$$

$$c' = c_{y_1} + c_{y_2} \quad (7)$$

$$a' = a_{y_1} \times a_{y_2} \quad (8)$$

D. Conditional Pattern

Definition 5. Exclusive Vertices *(see Figure 3(c)): for G = (V, A), $\forall v_x, v_{y_1}, v_{y_2} \in V$, if $\exists(v_x - v_{y_1}, p_1) \in A \wedge \exists(v_x - v_{y_2}, p_2) \in A$, then v_{y_1} and v_{y_2} are Exclusive Vertices. Vertices v_x, v_{y_1} and v_{y_2} form a composition of Conditional Pattern.*

In order to keep the execution probability of each path of the composite service in the final result, we do not calculate the QoS and transition probabilities for Conditional Patterns, which means that the QoS values and transition probability of each vertex in Conditional Pattern will stay the same.

E. Loop Pattern

1) General Loop:

Definition 6. Cyclic Vertices *(see Figure 3(d)): for G = (V, A), $\forall v_1, v_2, \ldots, v_n \in V$, if $\exists(v_1 - v_2, p_1) \in A \wedge \exists(v_2 - v_3, p_2) \in A \wedge \ldots \wedge \exists(v_{n-1} - v_n, p_{n-1}) \in A \wedge \exists(v_n - v_1, p_n) \in A$, then $v_1, v_2, \ldots,$ and v_n are Cyclic Vertices within a Loop. Vertices $v_1, v_2, \ldots,$ and v_n form a composition of Loop Pattern.*

Figure 3(d) is a Loop Pattern which starts from v_1 and may leave the Loop from either of v_1, v_2,..., and v_n with a certain probability indicated in the figure. The relationship of these probabilities is as follows: $p_i + \sum_{j=1}^{m_i} p_{ij} = 1 \quad (i \in [1, n])$. If vertex v_i ($i \in [1, n]$) is just an internal vertex of the Loop, i.e. $deg^+ v_i = 1$, then $p_{ij} = 0$ ($j \in [1, m_i]$). The cost of v_i is c_i. The availability of v_i is a_i.

After it has been executed for l ($l \in [0, +\infty)$) times, the Loop will be left from either of v_i ($i \in [1, n]$) to its adjacent vertex that does not belong to the Loop. Based on this description, Loop Pattern can be transformed into an equivalent graph but without the Loop (see Figure 4). In Figure 4, the combination of vertices v_1, v_2, ..., and v_n (the part within the dashed ellipse) is equivalent to one time of execution of the Loop in Figure 3(d). After the combination of vertices has been executed for l ($l \in [0, +\infty)$) times, finally, it will transfer from vertex v_i ($i \in [1, n]$) to its adjacent vertex according to the probability p_{ij} ($j \in [1, m_i]$).

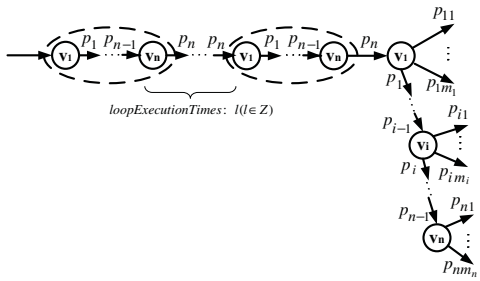


Figure 4. Loop Transformation

The probability for the combination of vertices to be executed for l ($l \in [0, +\infty)$) times and transferring from vertex v_1 to its adjacent vertex is $(\prod_{i=1}^{n} p_i)^l p_{1j}$ ($l \in [0, +\infty)$, $j \in [1, m_1]$); transferring from vertex v_k ($k \in [2, n]$) to its adjacent vertex is $(\prod_{i=1}^{n} p_i)^l (\prod_{i=1}^{k-1} p_i) p_{kj}$ ($l \in [0, +\infty)$, $j \in [1, m_k]$). Let $p_0 = 1$, the transition probability from v_k ($k \in [1, n]$) to its adjacent vertex is $(\prod_{i=1}^{n} p_i)^l (\prod_{i=0}^{k-1} p_i) p_{kj}$ ($l \in [0, +\infty)$, $j \in [1, m_k]$). Therefore, statistically, the transition probability from a Cyclic Vertex

to its adjacent vertex that does not belong to the Loop is:

$$p'_{kj} = \sum_{l=0}^{+\infty}\left(\prod_{i=1}^{n} p_i\right)^l \left(\prod_{i=0}^{k-1} p_i\right) p_{kj} = \frac{\left(\prod_{i=0}^{k-1} p_i\right) p_{kj}}{1-\prod_{i=1}^{n} p_i}$$
$$(k \in [1,n],\, j \in [1,m_k]) \tag{9}$$

The probability for the combination of vertices v_1, v_2, $\ldots$, v_n to be executed for l ($l \in [0,+\infty)$) times, transferring to vertex v_k ($k \in [1,n]$), and leaving the loop from v_k is $(\prod_{i=1}^{n} p_i)^l(\prod_{i=0}^{k-1} p_i)(1-p_k)$. At this time, the cost generated by all the Cyclic Vertices is $l\sum_{i=1}^{n} c_i + \sum_{i=1}^{k} c_i$ and the availability is $(\prod_{i=1}^{n} a_i)^l \prod_{i=1}^{k} a_i$. Therefore, statistically, the cost (c') and availability (a') generated by all the Cyclic Vertices within the Loop Pattern are as follows:

$$\begin{aligned} c' &= \sum_{k=1}^{n}\sum_{l=0}^{+\infty}\left(\left(\prod_{i=1}^{n} p_i\right)^l \left(\prod_{i=0}^{k-1} p_i\right)(1-p_k)\left(l\sum_{i=1}^{n} c_i + \sum_{i=1}^{k} c_i\right)\right) \\ &= \sum_{k=1}^{n} \frac{\left(\prod_{i=0}^{k-1} p_i\right)(1-p_k)\left(\sum_{i=1}^{k} C_i + \prod_{i=1}^{n} p_i \sum_{i=k+1}^{n} C_i\right)}{\left(1-\prod_{i=1}^{n} p_i\right)^2} \end{aligned} \tag{10}$$

$$\begin{aligned} a' &= \sum_{k=1}^{n}\sum_{l=0}^{+\infty}\left(\left(\prod_{i=1}^{n} p_i\right)^l \left(\prod_{i=0}^{k-1} p_i\right)(1-p_k)\left(\prod_{i=1}^{n} a_i\right)^l \prod_{i=1}^{k} a_i\right) \\ &= \sum_{k=1}^{n} \frac{\left(\prod_{i=0}^{k-1} p_i\right)(1-p_k)\prod_{i=1}^{k} a_i}{1-\prod_{i=1}^{n}(p_i a_i)} \end{aligned} \tag{11}$$

All the vertices in Figure 4 can be denoted by one vertex v' in Figure 5. The calculation for the transition probability, cost, and availability of vertex v' is based on Formulae 9, 10, and 11 respectively.

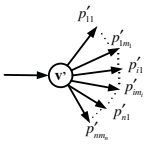


Figure 5. Loop Transformation Result

Formulae 9, 10, and 11 are general formulae for all types of Loop Patterns. Next, we will demonstrate how to use them to calculate the probability, cost, and availability for specific Loop Patterns.

2) Specific Loops:

- Self-Loop

If there is only one vertex in a Loop Pattern, the Loop is a self-loop (see Figure 6). Formally,

Definition 7. Self-Loop : *for a Loop Pattern defined in Definition 6, if $n = 1$, i.e. $\exists(v_1 - v_1, p_1) \in A$, then Loop Pattern of this kind is called Self-Loop.*

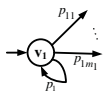


Figure 6. Self-Loop

For a Self-Loop, $k = 1$ for Formulae 9, 10, and 11. There are:

$$p'_{1j} = \frac{p_{1j}}{1-p_1} \quad (j \in [1,m1]) \tag{12}$$

$$c' = \frac{(1-p_1)c_1}{(1-p_1)^2} = \frac{c_1}{1-p_1} \tag{13}$$

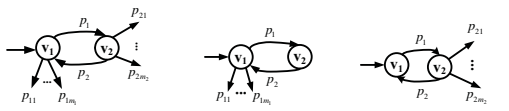
$$a' = \frac{(1-p_1)a_1}{1-p_1 a_1} \tag{14}$$

Self-Loop can be replaced by a single vertex (see Figure 5). The probability, cost, and availability for this vertex are calculated according to Formulae 12, 13, and 14.

- Dual-Vertex-Loop

If there are two vertices in a Loop, the Loop is a dual-vertex-loop (see Figure 7). Formally,

Definition 8. Dual-Vertex-Loop: *for a Loop Pattern defined in Definition 6, if n=2, i.e. $\exists(v_1 - v_2, p_1) \in A \wedge \exists(v_2 - v_1, p_2) \in A$, then this Loop Pattern is a Dual-Vertex-Loop.*



(a) General (b) Same-in-out (c) Different-in-out

Figure 7. Dual-Vertex-Loop

Figure 7(a) is the general form of a Dual-Vertex-Loop with $k = 2$ for Formulae 9, 10, and 11. There are:

$$p'_{1j} = \frac{p_{1j}}{1-p_1 p_2} \quad (j \in [1,m1]) \tag{15}$$

$$p'_{2j} = \frac{p_1 p_{2j}}{1-p_1 p_2} \quad (j \in [1,m2]) \tag{16}$$

$$c' = \frac{(1-p_1)(c_1+p_1p_2c_2)+p_1(1-p_2)(c_1+c_2)}{(1-p_1p_2)^2} = \frac{c_1+p_1c_2}{1-p_1p_2} \tag{17}$$

$$a' = \frac{(1-p_1)a_1 + p_1(1-p_2)a_1a_2}{1-p_1p_2a_1a_2} \tag{18}$$

In Figure 7(b), only the start vertex v_1 of the Loop can leave the Loop, i.e.

Definition 9. Dual-Vertex-Loop with the same in and out vertex: *for a Dual-Vertex-Loop, if $deg^+(v_1) > 1 \wedge deg^+(v_2) = 1$ then it is a Dual-Vertex-Loop with the same in and out vertex.*

For a Dual-Vertex-Loop with the same in and out vertex, $k = 2$, $p_{2j} = 0$ and $p_2 = 1$ for Formulae 9, 10, and 11. There are:

$$p'_{1j} = \frac{p_{1j}}{1-p_1p_2} = \frac{p_{1j}}{1-p_1} \quad (j \in [1,m1]) \tag{19}$$

$$c' = \frac{c_1+p_1c_2}{1-p_1p_2} = \frac{c_1+p_1c_2}{1-p_1} \tag{20}$$

$$a' = \frac{(1-p_1)a_1 + p_1(1-p_2)a_1a_2}{1-p_1p_2a_1a_2} = \frac{(1-p_1)a_1}{1-p_1a_1a_2} \tag{21}$$

A Dual-Vertex-Loop with the same in and out vertex can be replaced by a single vertex (see Figure 5). The probability, cost, and availability for this vertex are calculated according to Formulae 19, 20, and 21.

In Figure 7(c), only vertex v_2 can leave the Loop, i.e.

Definition 10. Dual-Vertex-Loop with different in and out vertices: *for a Dual-Vertex-Loop, if* $deg^+(v_1) = 1 \bigwedge deg^+(v_2) > 1$ *then it is a Dual-Vertex-Loop with different in and out vertices.*

For a Dual-Vertex-Loop with different in and out vertices, $k = 2$, $p_{1j} = 0$, and $p_1 = 1$ for Formulae 9, 10, and 11. There are:

$$p'_{2j} = \frac{p_1 p_{2j}}{1 - p_1 p_2} = \frac{p_{2j}}{1 - p_2} \quad (j \in [1, m2]) \tag{22}$$

$$c' = \frac{c_1 + p_1 c_2}{1 - p_1 p_2} = \frac{c_1 + c_2}{1 - p_2} \tag{23}$$

$$a' = \frac{(1 - p_1)a_1 + p_1(1 - p_2)a_1 a_2}{1 - p_1 p_2 a_1 a_2} = \frac{(1 - p_2)a_1 a_2}{1 - p_2 a_1 a_2} \tag{24}$$

A Dual-Vertex-Loop with different in and out vertices can be replaced by a single vertex (see Figure 5). The probability, cost, and availability for this vertex are calculated according to Formulae 22, 23, and 24.

III. QoS Analysis

In this section, we will discuss how the QoS probability distribution of a composite service is obtained.

In order to get every unique path of the composite service, a Service Graph needs to be transformed into a rooted tree. Before this transformation, all the composition patterns that do not belong to a tree structure, such as Parallel and Loop Pattern, will be replaced by equivalent vertices. Sequential Patterns will also be replaced by equivalent vertices to simplify the transformation process. The original composition patterns are recorded in their replacing vertices 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. After a rooted tree is obtained, the execution probability and QoS of different paths can be calculated. A histogram will then be presented to illustrate the QoS probability distribution for the composite service.

The process of QoS analysis is summarized as follows:

A. Sequential Pattern Replacement

All the Sequential Vertices within a Sequential Pattern will be replaced by a single vertex. The IDs of Sequential Vertices will be recorded in the replacing vertex so that the original information of the path will not be lost. The calculation of the replacing vertex's transition probability and QoS is based on Formulae 3, 4, and 5. The process of replacing Sequential Patterns is shown in Algorithm 1[1].

```
set currV to the head vertex of the table;
while currV has next vertex in the table do
    if outdegree of currV is one then
        set succV to the direct successor vertex of currV;
        if currV and succV is sequential-connected AND indegree of
        succV is one then
            set probability of currV to probability of succV;
            compute cost of currV as cost of currV + cost of succV;
            compute availability of currV as availability of currV ×
            availability of succV;
            set ID of currV to ID of currV '&' ID of succV;
            set direct successor vertices of currV to direct successor
            vertices of succV;
            remove succV from the table;
        else
            set currV to the next vertex of currV in the table;
        end
    else
        set currV to the next vertex of currV in the table;
    end
end
```

Algorithm 1: Sequential Pattern Replacement

B. Loop Pattern Replacement

All the Cyclic Vertices within a Loop Pattern will be replaced by a single vertex. The IDs of them will be recorded in the replacing vertex. The calculation of the replacing vertex's transition probability and QoS is based on Formulae 9, 10, and 11. The process of replacing Loop Patterns is shown in Algorithm 2[2].

```
call FindLoopInServiceGraph(G = (V, A));
for each loop do
    set newV to the start vertex of loop;
    set ID to NIL;
    for each vertex in loop do
        store the probability of each arc belonging to loop in ProbList;
        store the cost of each vertex in CostList;
        store the availability of each vertex in AvailList;
    end
    set count to zero;
    for each vertex in loop do
        if outdegree of vertex is more than one then
            for each outgoing arc of vertex except the arc belonging
            to loop do
                call ArcProbCalc(probability of arc, count,
                ProbList);
                add vertex's successor vertex connected by this arc
                to newV's direct successor vertex;
            end
        end
        set ID to ID '!' ID of vertex;
        if vertex is not newV then
            remove vertex from loop;
        end
        increment count;
    end
    call QoSCalculation(count, ProbList, CostList, AvailList);
    set the cost of newV to the return of cost of QoSCalculation;
    set the availability of newV to the return of availability of
    QoSCalculation;
    set the ID of newV to ID;
end
```

Algorithm 2: Loop Pattern Replacement

[1]all the vertices of the Service Graph are stored in a table. Each vertex has a unique ID. The source vertex (indegree=0) of the Service Graph is the first element of the table while the sink vertex (outdegree=0) is the last element of the table.

[2]to simplify the process of discovering loop (cycle) in graph, arcs within loop that connect Cyclic Vertices together are marked with special symbols just like arcs that are sequential-connected using '-' or concurrent-connected using '=' to represent the connection way.

C. Parallel Pattern Replacement

All the Parallel Vertices within a Parallel Pattern will be replaced by a single vertex. The IDs of them will be recorded in the replacing vertex. The calculation of the replacing vertex's transition probability and QoS is based on Formulae 6, 7, and 8. The process of replacing Parallel Patterns is shown in Algorithm 3.

```
set currV to the head vertex of the table[1];
while currV has next vertex in the table do
    read the number of currV's concurrent-connected vertices;
    if currV has concurrent-connected vertices then
        set probability of newV to one;
        set cost of newV to zero;
        set availability of newV to one;
        set ID of newV to NIL;
        set the direct successor vertex of currV to newV AND the
        connection way to sequential-connected;
        set the direct successor vertex of newV to the direct successor
        vertex of currV's concurrent-connected vertex AND the
        connection way to sequential-connected;
        for each concurrent-connected vertex of currV do
            compute cost of newV as cost of newV + cost of
            concurrent-connected vertex;
            compute cost of newV as cost of newV + cost of
            concurrent-connected vertex;
            compute availability of newV as availability of newV ×
            availability of concurrent-connected vertex;
            set ID of newV to ID of newV '*' ID of
            concurrent-connected vertex;
            remove concurrent-connected vertex from table;
        end
    end
    set currV to the next vertex of currV in the table
end
```

Algorithm 3: Parallel Pattern Replacement

D. Transformation from a Service Graph to a Rooted Tree

After the processes of Sequential, Parallel and Loop Pattern replacement, a Service Graph now is ready to be transformed into a rooted tree. The detailed transformation process is shown in Algorithm 4.

First, the statuses of all the vertices in the Service Graph are marked as unaccessed. The transformation process starts from the source vertex of the Service Graph (Line 1 to 4). The process will keep visiting the current vertex's next unaccessed vertex as the current vertex till the sink vertex of the Service Graph is reached (Line 5 to 20). This is called Sequential Access Process. All the vertices that have been visited during Sequential Access Process are recorded in the rooted tree as a branch of the tree, with the previously visited vertex being the parent of the followingly visited vertex. Then this branch is revisited in a reverse order which is called Inverse Access Process (Line 21 to 28). Whenever a vertex with multiple outgoing edges (outdegree>1) is reached, the Inverse Access Process will stop. Another Sequential Access Process will be invoked. Then Inverse Access Process and Sequential Access Process will be alternatively invoked until all the vertices in the Service Graph have been accessed. Finally, a rooted tree is obtained.

```
1  set currV to the head vertex of the table[1];
2  store currV as the root in RootedTree;
3  set currLeaf to the root of RootedTree;
4  set status of all vertices in the table to UNACCESSED;
5  while true do
6      if outdegree of currV is zero then
7          store currV as child of currLeaf in RootedTree;
8          set currV to FindVertexWithSameIDInTable(currLeaf);
9      else
10         set count to zero;
11         for each direct successor vertex of currV do
12             if status of this vertex is UNACCESSED then
13                 store this vertex as child of currLeaf in RootedTree;
14                 set currLeaf to child of currLeaf;
15                 set currV to this vertex;
16                 set status of this vertex to ACCESSED;
17                 increment count;
18                 break from for;
19             end
20         end
21         if count is zero then
22             if indegree of currV is zero then
23                 break from while
24             else
25                 set CurrLeaf to parent of currLeaf in RootedTree;
26                 set currV to
                   FindVertexWithSameIDInTable(currLeaf);
27             end
28         end
29     end
30 end
```

Algorithm 4: Rooted Tree Generation

E. Paths of a Composite Service

Each path of the composite service will be extracted from the rooted tree. Then the execution probability, execution conditions, and QoS for each path can be obtained.

Since every vertex except for the root has only one parent in the rooted tree, we can keep track of each branch of the tree by visiting from every leaf of the tree backward to the root. This branch is a path of the composite service in its reverse order. After reversing the order, one path of the composite service is obtained.

The detailed process of extracting paths from the rooted tree is shown in Algorithm 5.

```
set pathNumber to one;
for each leaf of RootedTree do
    set currV to leaf;
    while true do
        store currV in ReversePathList[pathNumber];
        if currV does not have parent then
            break from while;
        else
            set currV to parent of currV;
        end
    end
    store vertices of ReversePathList[pathNumber] in reverse order in
    PathList[pathNumber];
    increment pathNumber;
end
```

Algorithm 5: Path Extraction

F. QoS Probability Distribution

For each path obtained by Algorithm 5, it is seen as a Sequential Pattern. The calculation of cost and availability for each path is based on Formulae 4 and 5. The execution

probability for each path is the product of the transition probability of every arc in the path.

The IDs of the vertices within each path record the actual conditions of taking each path: the original vertices of each path and their relationships (i.e. different composition patterns).

The result will be shown in a histogram with each bar representing an execution path of a composite service, the width of the bar representing the probability of taking this path, and the height of the bar representing the QoS of this path. Other QoS information, such as the mean (which is calculated as the mathematical expectation of the QoSs of all the paths), maximum, and minimum QoS values of executing the composite service, will also be shown in the histogram.

IV. A Running Example

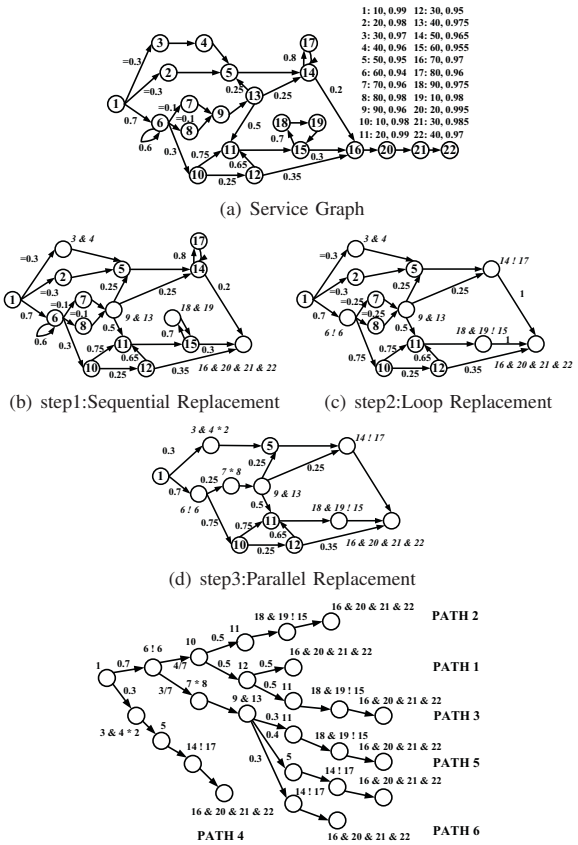
Figure 8(a) is the Service Graph of a composite service. It includes the basic patterns that are discussed in Section II. Each vertex in it has a unique ID. The transition probability, with a default value 1, is marked beside the arc. '=' represents concurrent-connected way between vertices. The default way is sequential-connected. One execution plan is given on the right side of the figure. The ID of each vertex is followed by the cost and availability values of it. For example, $12 : 30, 0.95$ represents that the cost is 30 and the availability is 0.95 for the Web service executing task 12 (represented by the vertex with ID 12 in the graph) of the composite service.

Symbols '&', '*', and '!' are used to represent vertices coming from Sequential, Parallel, and Loop Patterns respectively. For example, after Sequential Pattern replacement, vertices 3 and 4 in Figure 8(a) are replaced by vertex 3&4 in Figure 8(b).

According to Section III, the process to get the QoS probability distribution for the composite service is as follows:

- After the first five steps (shown in Figure 8), a Service Graph is transformed into a rooted tree and each path of the composite service is obtained. The ID of all the paths are marked out in Figure 8(e).
- At the sixth step, the execution probability and QoS of each path are calculated. The resulting QoS probability distribution histograms are shown in Figure 9.

Now, it can be seen from Figure 9 that there are seven paths in the composite service. The path number is referred to the path ID in Figure 8(e). The mean cost and availability for this process are 897.14 and 0.58 respectively which are represented by horizontal straight lines. The highest cost is 1220.00 generated by path 7 with the lowest availability of 0.491 and a probability of 0.044. The lowest cost is 360 generated by path 1 with the highest availability of 0.733 and a probability of 0.046. The cost of path 5, 6 and 7 is higher than the mean cost, and the cost of path 1, 2, 3 and 4 is lower than the mean cost. The probability is 0.175 (0.087+



(a) Service Graph

(b) step1:Sequential Replacement

(c) step2:Loop Replacement

(d) step3:Parallel Replacement

(e) step4,5:Tree and Paths Generation

Figure 8. QoS Analysis Process

$0.044 + 0.044$) for the process to generate a cost higher than the mean cost, and is 0.825 (0.046 + 0.394 + 0.085 + 0.3) for the process to generate a cost lower than the mean cost. Availability information can be analyzed in a similar way.

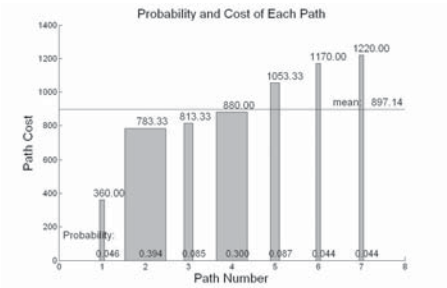
V. Related Work

In this section, we shall look into different methods for calculating the basic composition patterns.

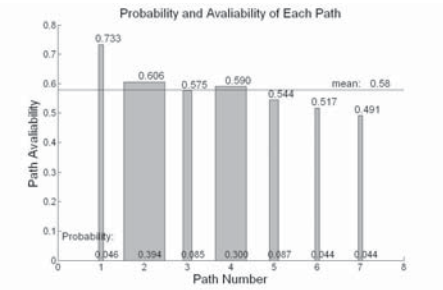
For sequential patterns, the calculation method is the same in all kinds of QoS calculation methods.

For parallel patterns, [6] uses the same calculation method as what is used for sequential structures except for the calculation of time related attributes such as response time and execution time. In [6], time (response time or execution time) is the maximum time of all the parallel tasks. For conditional structures, [6] uses a statistical method to calculate the QoS. [4], [9] and [7] follow [6] to calculate the QoS for parallel and conditional structures.

In [12], the path with the maximum execution duration in parallel and conditional structures is called the critical path. The execution duration of this critical path is the duration of the whole process. The calculation of availability and successful execution rate is based on the critical path. The



(a) Cost



(b) Availability

Figure 9. QoS Probability Distribution

limitation of selecting Web services based on the QoS of the critical path has been discussed in [2]. In [3], the QoS of every path of the composite service is calculated and guaranteed to satisfy the user specified requirements.

For loop patterns, [6] reduces loop patterns by recalculating the outgoing transition probability of the loop as well as the QoS of the nodes in the loop. [6] only provides the solution for two types of loop patterns: Self-Loop and Dual-Vertex-Loop with the same in and out vertex. [12] unfolds the loop patterns by determining the maximum number of times that the loop is taken. If a loop is taken a maximum of n times, then n+1 copies of the nodes in the loop pattern will appear in the resulting acyclic process. [4] follows this method to calculate the QoS for loop patterns. [9] does not deal with loop patterns.

In [8], the QoS of each type of patterns is calculated on a maximum and minmum basis. For example, the maximum and minimum execution time for conditional pattern is the execution time of the path with the maximum and minimum execution time respectively, while the maximum and minimum execution time for parallel pattern is the same which is the execution time of the path with the maximum execution time.

Our approach extends existing work by formally defining those above mentioned composition patterns and providing a QoS calculation solution for all types of loop patterns.

VI. Conclusion

The approach proposed in this paper goes beyond the existing QoS analysis methods by providing a QoS probability distribution with both the execution probability and execution conditions of each path in a service composition. The QoS analysis result shown in a histogram explicitly presents the maximum, mean, minimum QoS of a service composition and QoS execution probabilities for individual paths. Therefore the proposed approach for QoS Analysis can provide users/designers with more detailed information about QoS of a service composition.

In this paper, the QoS probability distribution is calculated based on the assumption that each task has a fixed QoS. The QoS of each task is more likely to be a continuous probability distribution in reality. As the next step of research, we will study the general method of QoS analysis for a service composition based on a continuous QoS probability distribution of tasks.

References

[1] Qos for web services: Requirements and possible approaches. Technical Report 25, W3C Working Group, November 2003.

[2] D. Ardagna and B. Pernici. Global and local qos guarantee in web service selection. In *Workshop on Business Processes and Services 2005*, volume 3812/2006, pages 32–46. LNCS, 2006.

[3] D. Ardagna and B. Pernici. Adaptive service composition in flexible processes. *Software Engineering, IEEE Transactions on*, 33(6):369–384, June 2007.

[4] 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: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1069–1075, New York, NY, USA, 2005. ACM.

[5] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani. Qos-aware replanning of composite web services. In *ICWS '05: Proceedings of the IEEE International Conference on Web Services*, pages 121–129, Washington, DC, USA, 2005. IEEE Computer Society.

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

[7] S.-Y. Hwang, H. Wang, J. Tang, and J. Srivastava. A probabilistic approach to modeling and estimating the qos of web-services-based workflows. *Inf. Sci.*, 177(23):5484–5503, 2007.

[8] M. Jaeger, G. Rojec-Goldmann, and G. Muhl. Qos aggregation for web service composition using workflow patterns. *Enterprise Distributed Object Computing Conference, 2004. EDOC 2004. Proceedings. Eighth IEEE International*, pages 149–159, Sept. 2004.

[9] D. Menasce. Composing web services: A qos view. *Internet Computing, IEEE*, 8(6):88–90, Nov.-Dec. 2004.

[10] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: A research roadmap. *International Journal of Cooperative Information Systems*, 17(2):223–255, 2008.

[11] Q. Yu and A. Bouguettaya. Framework for web service query algebra and optimization. *ACM Trans. Web*, 2(1):1–35, 2008.

[12] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. Qos-aware middleware for web services composition. *Software Engineering, IEEE Transactions on*, 30(5):311–327, May 2004.