Journal of Computers Vol. 28, No. 1, 2017, pp. 149-166
doi:10.3966/199115592017022801012

QBand: Indicating the Implementation Distribution of QoS-based Web Service Composition Solutions

Gang Wang[1], Zhen-Zhong Zhang[2]

[1] Department of Computer Science and Technology, Tongji University
Shanghai
f_lag@buaa.edu.cn

[2] Institute of Information Engineering, Chinese Academy of Sciences
Beijing

Received 16 September 2015; Revised 30 March 2016; Accepted 16 April 2016

Abstract. Considerable researches focus on the improvement of the algorithms of service composition while ignoring the flexibility of users to set QoS constraints. Most of them assume that users could specify QoS constraints easily. But in reality, there are more considerations in setting QoS constraints. One of them is that a user wants to know the QoS situation of service composition solutions which implement the user's task based on candidate web services available in the market. Then he can consider both QoS requirements expectations and actual QoS implementation levels to set QoS constraints. So we propose a data structure QBand to provide more information on QoS implementation levels of current web service compositions implementing the user's task. It calculates two aspects of information: 1) the QoS boundary of the service composition solutions implementing the task based on QoS data of candidate web services, 2) the numbers of the service compositions available for each value in QoS boundary. QBand provides users an overview of current QoS implementation levels of service composition solutions for the users' task, to assist users during setting QoS constraints. The algorithm can also be used for quantitive attributes of other composite system.

Keywords: QoS implementation levels, requirements engineering, service composition, statistical description

1 Introduction

SOA (Service-Oriented Architecture) is a popular development paradigm for information systems, by coupling various web services loosely and quickly to alter business processes of enterprises to meet new requirements. Web services as a basic technology for SOA have been more and more used. With continually increasing web services provided on the Internet, functionally similar web services inevitably emerge, and NFRs (Non-Functional Requirements), often called QoS (Quality of Service), play a key factor in web service selection. A user's task may have multiple sub-tasks, and each sub-task could be implemented by multiple web services, called candidate web services for a sub-task. For one from the candidate web services may be better than the other in one or more QoS attributes or vice versa, there is not a unique solution by selecting from these candidate web services for each sub-task and composing of them to implement the user's task. So the user's QoS constraints and preferences on the task are added to differentiate between solutions. Considering QoS, web service composition is a MCDM (Multiple-Criteria Decision Making) problem, and the criteria herein are QoS attributes, some attributes of which may conflict such as cost and performance [1]. The cheapest composition solution of web services may usually not have the best performance. It needs to tradeoff among QoS attributes, so finding a solution of web service composition to satisfy the user's QoS requirements is an optimization problem, called QoS-based web service composition.

The process of QoS-based web service composition contains two parts: one is that users set QoS requirement constraints, and the other is that service composition algorithms search for the optimization solution to meet QoS constraints. The setting of QoS constraints affects whether the service composition algorithms can find the solution the users want. But current research in QoS-based web service composition focus on web service composition algorithms 0-0, due to the complexity of the combinational optimization problem caused by a large number of web services with different QoS characteristics available. Most of them support QoS constraints, but weak facilities are provided to help users to set QoS constraints. There are considerations in setting QoS constraints in reality. One of them is that a user wants to know the QoS situation of service composition solutions which implement the user's task based on candidate web services in the market. Then he can consider both QoS requirements expectations and actual QoS implementation levels to set QoS constraints.

The above consideration comes from three following situations. First, consider a user wants a service composition of response time for a task less than 50 milliseconds. It may happen that no service composition solution is found if the implementation level of response time of current service composition on the web has not reach lower than 50 milliseconds. What's the lowest response time of current service composition on the web for the task? The user wants to know whether the lowest response time is acceptable. If it is acceptable, the user adjusts his QoS constraints according to current actual QoS implementation levels for getting a service composition solution. Second, consider a user imposes a constraint on response time for a task less than 1500 milliseconds. The response time implementation level of the solution found by web service composition algorithms may not be the cost-effective mainstream implementation level although the solution might have the other best QoS attributes such as the lowest cost. What's the range of the mainstream implementation level of response time of current service composition for the task? The user doesn't know the range. Third, the user may have ambiguous QoS requirements [7], he may just want the composition of web services in one QoS implementation level such as in low QoS, medium QoS or high QoS according to his economics. The situation is noticeable in the case of widely used services, such as hotel booking, weather forecast services. But the user doesn't know what ranges these QoS implementation levels are, thus doesn't set the QoS constraints.

All in all, all these are caused by that the user is not clear about QoS implementation levels of current service compositions. It is required that a facility to provide the information on QoS implementation levels of current service composition to assists the users to set their QoS constraints according to their QoS requirements expectations, combined with QoS implementation levels of service compositions available on the Internet. It needs the calculation of QoS and its corresponding number of service compositions implementing the users' tasks. However, there is the exhaustive method that is the only and primary method to fulfil the function currently. The exhaustive method takes much time. A better method needs the development.

To this end, we develop a data structure named QBand for providing more information in respect of QoS implementation levels of current web service compositions implementing the user's task. It includes two aspects of information: 1) the QoS boundary of the service composition solutions implementing the task based on QoS data of candidate web services, 2) the numbers of the service compositions available for each value in QoS boundary. Further, we design a kind of spectrogram to visually represent the implementation distribution of the service composition solutions. QBand provides the users an overview of current QoS implementation levels of service composition solutions for the user's task such as the mainstream QoS implementation level according to the implementation distribution to assist users during setting QoS constraints. The algorithm can also be used for indicating quantitive attributes of other composite system.

The remainder of this paper is organized as follows. Section 2 describes the model of QoS-based web service composition. Section 3 details the data structure QBand. At the beginning of Section 3, it describes the definition and semantics of QBand. Then Section 3 describes QBand operations, its appearance and evaluates it. Section 4 illustrates how QBand works and evaluate it. Section 5 presents our prototype implementation to show the scenario using QBand. Section 6 discusses the related work. Finally, Section 7 concludes.

2 Model Description of QoS-Base Web Service Composition

This paper uses the symbols in Fig. 1 to represent the execution logic of tasks. In Fig. 1, t_i represents a task to complete. There are four basic execution logic: sequence, switch, parallel and loop. Most of complex execution logic can be implemented through the combinations of four basic execution logic. The parameter k represents the estimated number of iterations in a loop construct determined by the way in [1, 8]. The parameter u used as a subscript represents the first task in the loop, and v used as a subscript represents the last task in the loop.

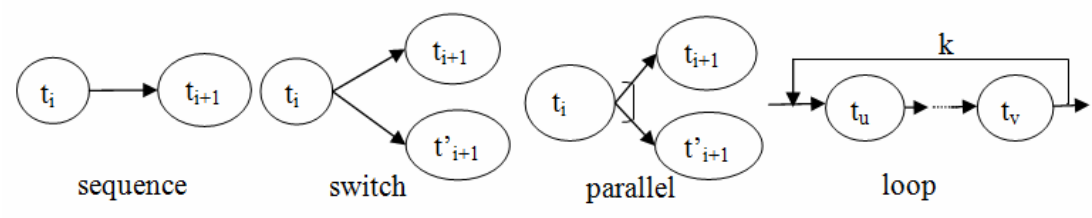


Fig. 1. Four basic execution logic

Fig. 2 uses the symbols in Fig. 1 to represent an example of a complex task. This complex task includes 9 sub-tasks t_1, t_2, ..., t_9 which covers four basic execution logic.

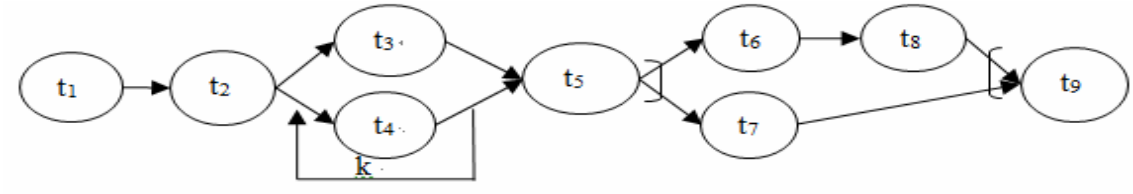


Fig. 2. An example of combination to fulfill a complex task

There are several candidate web services for each task t_i. Candidate web services can complete the task t_i. The difference between them is non-functional/QoS attributes, such as response time and cost.

Web service composition is shown in Fig. 3. Assuming a complex task, marked as T, requires m subtasks to finish labeled t_1, t_2, ..., t_m. Each sub-task t_i, has a n_i corresponding web service candidates, labelled $ws_{i,1}$, $ws_{i,2}$, ,..., and ws_{i,n_i}.

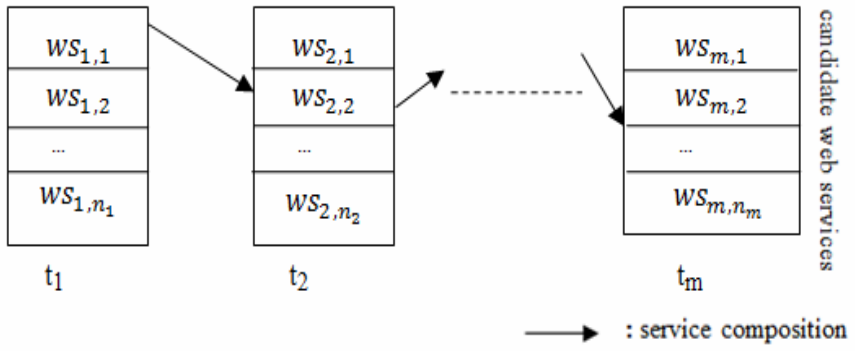


Fig. 3. The process of web service composition: Each task t_i has n_i candidate web services. Choose one candidate web service for each task, to accomplish complex tasks T

To complete the complex task T, we need to select a candidate web service for each sub-task. A web service composition solution is called an execution plan. For example, the web service composition solution ($ws_{1,1}$., $ws_{2,1}$,..., $ws_{m,1}$) is an execution plan of the complex task T.

For a complex task, there may be many web service composition solutions. Each web service composi-

tion solution has a different QoS characteristics obtained from the component web services. Selection of composition solutions has become a problem. Currently, there are several approaches for QoS-aware web service composition. But most approaches are concerned about web service composition algorithm itself, while ignoring the flexibility for users to set QoS. Most of them require QoS constraints given in form of numbers. They are based on a hypothesis: the user can explicitly give or describe their non-functional requirements or QoS. But in fact, the hypothesis is weak because of the vagueness of users about non-functional requirements on their tasks and the lack of the information about the QoS of web services on the market.

3 QBand: Indicating QoS Implementation Levels of Service Composition Solutions for the Task

This section is the main part of this paper including three subsections. To indicate the QoS implementation levels of web service composition solutions for a task, Subsection 3.1 introduces the functions and definition of QBand. Subsection 3.2 introduces QBand operations to calculate QoS implementation levels. Finally, Subsection 3.3 introduces a kind of spectrogram to visually represent them, providing an overview of current QoS implementation levels to assist users during setting QoS constraints.

3.1 Functions and Definition of QBand

In this paper, we propose a data structure QBand (QoS Band) to indicate QoS levels of current web service compositions implementing the task. Derived from the requirements that the user is concerned about the QoS levels of current service compositions implementing the task, the QBand representing the implementation distribution of QoS-based web service composition solutions has the following two functions:

1. QoS Boundary of current web service compositions available to implement the task, including the minimum and maximum QoS values.
2. Numbers of current web service compositions available for each value in QoS boundary.

According to the two functions above, we develop QBand. It has two parts: the storage area and operations. The pseudo-code description of the QoS band is as follows. The storage area stores QoS information and numbers of candidate web services in the array named band. The QBand operations have two types: aggregation operations and incremental operations. The aggregation operations are defined to calculate composite QoS information and numbers of service composition solutions to implement the composite task. The incremental operations are defined to update QoS information and numbers of candidate web services when new web services for the task come, old web services close, or QoS of existing web services change

```
1 QBand
2{
   /*number each QoS value(qMin+i) can achieve; i is from 0 to the length
   of band(qMax-qMin+1)*/
3  integer band[];
   /* min QoS in candidate ws for a task*/
4  integer qMin;
   /* max QoS in candidate ws for a task*/
5  integer qMax;
  /*QBand aggregation operations:
    used to calculate QBand of the composite task. The implementation of
     QBand operations depends on execution logic and characteristics of QoS
     attributes. E.g. QBand + is used for the QoS attribute response time in
     sequence construct.*/
6   QBand + (QBand q1, QBand q2);
7   QBand ×(QBand q1, QBand q2);
8   QBand ×(float p, QBand q2);
9   QBand max(QBand q1, QBand q2);
10  QBand pow(integer k, QBand q1);
  /*QBand incremental operations:
    used to update the QBand of the task when candidate web services for
    the task change.*/
```

```
11    QBand +in (QBand q1, QBand q2);
12    QBand –in (QBand q1, QBand q2);
13 }
```

The semantic description of QBand is shown in Fig. 4. We use the array index to represent QoS values and the array to hold the achieving number of each QoS value, which makes data structure small and runs faster than in the form of the linklist. The first element of the QBand array is the number of service composition solutions, whose QoS attribute is the minimum value. The i^{th} element indicates the number of service compositions whose QoS attribute value is the minimum plus the index i^{th}. It is the same until the maximum QoS attribute value.

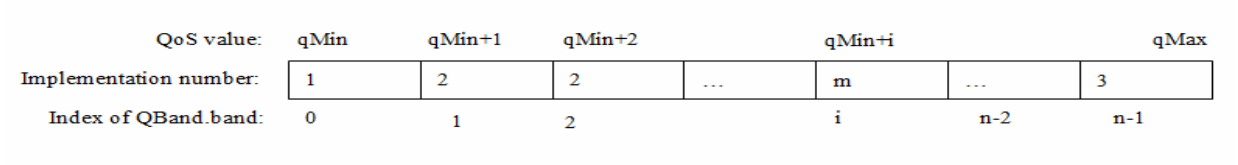


Fig. 4. Internal structure of QBand and its meaning

3.2 Operations of QBand

To fulfil the above two functions, on the one hand, we should get the QoS of the service composition solutions implementing the user's task. Candidate web services are published on the Internet by using UDDI (Universal Description Discovery and Integration). The composite QoS of the service composition solutions needs to be calculated based on QoS data of candidate web services available on the market. The calculation methods are adopted from the widely used methods in the works [1, 8] shown in Table 1. In Table 1, m represents the number of tasks in a sequence construct aforementioned in Section 2, p represents the number of parallel tasks in a parallel construct, p_i represents the probability of the case i in a switch construct where $\sum_{i=1}^{n} p_i = 1$, k represents the estimated number of iterations in a loop construct.

Table 1. QoS calculation of composite web services.

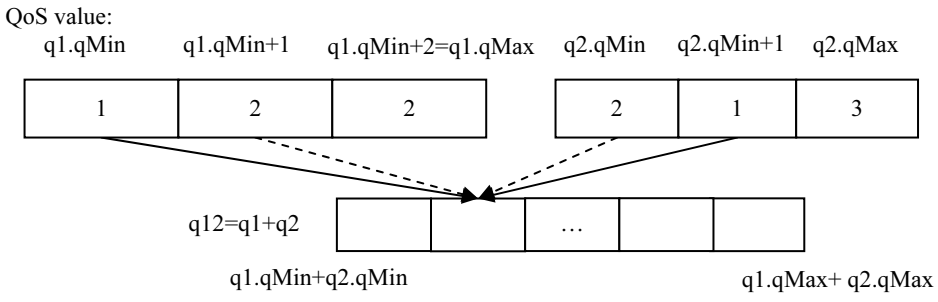
	sequence	parallel	switch	loop
response time	$\sum_{i=1}^{m} rt_i$	$Max_{i=1}^{p}(rt_i)$	$\sum_{i=1}^{n}(p_i * rt_i)$	k*rt
availability	$\prod_{i=1}^{m} a_i$	$\prod_{i=1}^{p} a_i$	$\sum_{i=1}^{n}(p_i * a_i)$	ak
reliability	$\prod_{i=1}^{m} r_i$	$\prod_{i=1}^{p} r_i$	$\sum_{i=1}^{n}(p_i * r_i)$	rk
cost	$\sum_{i=1}^{m} c_i$	$\sum_{i=1}^{p} c_i$	$\sum_{i=1}^{n}(p_i * c_i)$	k*c

On the other hand, we calculate the corresponding numbers of service composition solutions available of each value in QoS boundary. For the simple task which has no subtask, the corresponding number of each QoS value could be counted by candidate services implementing the task of this QoS value directly, but for the complex task with sub tasks, it should be solved that how the numbers of the service composition solutions are gotten based on the calculated numbers implementing the subtasks should be solved. The feature of execution logic of the complex task, characteristics of QoS attributes and realization on the above defined QBand array should be considered. To address it, we define five corresponding operations for QBand.

Because of the similarity of QBand operations, we just show addition of two vectors in QBand space to explain how to calculate composite QoS QBand. Its details are given in the following pseudo codes, and the working way of QBand + is generally described in the Process part at the beginning of the pseudo codes.

```
/*calculate the QBand of the composite task t12 based on the QBand q1 of
the sub-task t1 and the QBand q2 of the sub-task t2.
Process:
 1) calculate the QoS boundary implementing the task t12, create a new
    QBand of the task t12 (line2-4)
 2) calculate the numbers of the service composition in the QoS boundary
    (line5-9)
The principle of QBand + is shown in Fig. 5*/
1 QBand + (QBand q1, QBand q2) {
2    integer qMin = q1.qMin + q2.qMin;
3    integer qMax = q1.qMax + q2.qMax;
4    QBand q = new QBand(qMin, qMax);
5    for (integer i = 0; i < q1. band.length; i++) {
6        for (integer j = 0; j < q2. band.length; j++) {
7            q.band[i + j] += q1.band[i] * q2.band[j];
8         }
9    }
10     return q;
11 }
```

Example of QBand Addition: The operands in QBand addition are QoS properties which of composite web services are calculated by the way of summing shown in Table 1. Here we use Response Time (RT) as example. The value of RT is the time it takes to respond to a request for web service. Let q1 is the RT QBand of a task t1 and q2 is the response time QBand of the sequential task t2, then the range of the response time of the composite task of t1+t2 is [q1.qMin + q2.qMin, q1.qMax+q2.qMax]. In Fig. 5, q1.band [q1.qMin+1] means there are two candidate web services for t1 with a RT value of [q1.qMin+1]. The algorithm takes one element from q1.band to get the number of services with a RT value of [q1.qMin+i], and takes one element of q2.band to get the number of services with a RT value of [q1.qMin+j], then multiplies them up to get the number of service compositions with a RT value of [q1.qMin+i+q2.qMin+j] for the task t12. As shown in Fig. 5, it is calculated just by q1.band [i]* q2.band [j], and the number of service compositions with a RT value of [q1.qMin+i+q2.qMin+j] is just held in the q12.band [i+j]. This is partial number of service compositions with a RT value of [q1.qMin+i+q2.qMin+j]. We get all numbers by using two loops. The numbers are held in q12.band [i+j].



1) q12.band [q1.qMin+i+q2.qMin+j- (q1.qMin+q2.qMin)] =q12.band [i+j],
2) q12.band [i+j] +=q1.band [q1.qMin+i-q1.qMin]* q2.band [q2.qMin+j-q2.qMin],
3) q12.band [i+j] +=q1.band[i]*q2.band[j]

e.g. q12.band [1] =q1.band [0]*q2.band [1] +q1.band [1] + q2.band [0]

Fig. 5. Illustration of QBand addition. q12.band[1] has two sources: one is q1.band[0]*q2.band[1], the other is q1.band[1]*q2.band[0]

We define ***zero*** and ***unit elements*** in QBand space like real vector space:
qBand0 {band={1};qMin= qMax=0;} so that qBand+ qBand0= qBand
qBand1 {band={1};qMin= qMax=1;} so that qBand×qBand1= qBand

Because we use integral array index to represent the QoS, we scale them to deal with QoS attributes in form of float. For example, the availability of a web service is 0.95 we can scale 0.95 to 95. To deal with arising float during QBand operations, we round them. QBand ×(float p, QBand q2) is an example:

```
/* QBand × is used in the switch construct
  Process:
    1) calculate the QoS boundary of the task in the switch construct, cre-
    ate a new QBand for the task t12 (line2-4)
    2) calculate the numbers of the services for the task in the boundary
    (line5-8)*/
1 QBand ×(float p, QBand q1) {
    /* p represents the execution probability of the case in a switch con-
    struct. p * q1.qMin may be float, we round them to make sure the form
    of QoS is integer*/
2    integer qMin = Math.ceil(p * q1.qMin);
3    integer qMax = Math.ceil(p * q1.qMax);
4    QBand q = new QBand(qMin, qMax);
5    for (integer i = 0; i < q1.band.length; i++) {
6        integer index = Math.ceil(p * (q1.qMin + i));
7        q.band[index - qMin] += q1.band[i];
8    }
9    return q;
10 }
```

With the above defined QBand operations, we can calculate QBand of the task based on the calculated QBand of the subtasks. The calculation is a recursive process because a task can be composed of subtasks. First, we should define a data structure to describe the task. Although the definition of the task may differ in the implementation, its description contains the following information

```
1 Task {
    /*composite type (parallel, switch, loop) or simple type*/
2    Category;
    /*QoS data of candidate services implementing the task;
      statistic QoS and numbers of services are calculated from them.*/
3    wsq;
   /* executive probability of the branch task in the switch construct*/
4    p;
   /* loop number, p and k are determined by the way in [1, 8]./
5    k;
   /* the information about the branches contained in the parallel con-
    struct or switch construct*/
6    List branches;
    /* point to the successive task/
7   Task *next;
8 }
```

The recursive process of the QBand calculation is:

```
/* get the composite QBand of the task, we calculate QBand of each execu-
tion logic and accumulate them (line3-6)*/
1 QBand getCompositeQBand(Task t) {
2  QBand q = qBand0;
3  while (t != null) {
4    q = q+ getComponentQBand(t);
5    t = t.getNext();
6  }
7  return q;
8 }

/*according to the execution logic, use the corresponding function to get
QBand of the sub-task*/
1 QBand getComponentQBand (Task t) {
   /*get the category of the execution logic of the sub-task,
    according to the category of the execution logic, use the corresponding
    function to get QBand of the sub-task*/
2  if (t.getCategory()==X)) return getXQBand(t);
3  }
```

Because of the similarity of functions for the four execution logic, we just show getParallelQBand() and getQBand(), omitting other similar getXBand ().

```
/*used for accumulative QoS attributes, e.g. response time*/
1 QBand getParallelQBand(Task t) {
2   QBand q = qBand0;
    /*get parallel component tasks in composite task t;*/
3   List<Task> tList = t.getPTasks();
4   for (Task ptask to : tList) {
5   q = QBand.max(q, getCompositeQBand(ptask));
6   }
7   return q;
8 }

/* According to qos of candidate web services for the sample task t, form
the QBand of t*/
1 QBand getQBand(Task t) {
  /* get the min and max qos of candidate web services for t, initialize a
  new QBand according to the min and max qos*/
2   QBand q = new Q(t.getMinQoS(), t.getMaxQoS());
  /* get the numbers of the service compositions available for each value
  in QoS boundary and form the QBand of t*/
3   q.setBand(t.getWSQ());
4   return q;
5 }
```

Incremental calculation of QBand. When new web services for a task come, old web services close, or QoS of existing web services change, QBand for the task may change. To avoid recalculation from the beginning, we define internal addition and subtraction of QBand for a task to achieve incremental calculation. "Internal" means all the QBand operands representing the QoS implementation levels of the same task.

```
/*QBand +in is used to update the QBand of the task when new candidate web
services are provided.
 q1 is the QBand of the task, q2 is the QBand formed by the new web services
 for the task.
 Process:
 To update the QBand of the task,
   1) update the QoS boundary of the task, because the new minimum or maxi-
    mum QoS may exist when new web services are provided. Create a new QBand
    for the task (line2-4)
   2) accumulate the numbers of the services for the task in the QoS bound-
    ary (line5-12)*/
1 QBand +in (QBand q1, QBand q2){
2   integer qMin = q1.qMin < q2.qMin ? q1.qMin: q2.qMin;
3   integer qMax = q1.qMax > q2.qMax ? q1.qMax: q2.qMax;
4   QBand q = new QBand(qMin, qMax);
    /* add the numbers of the old web services and the numbers of the new
     web services according to the same QoS value*/
5   for (integer i = 0; i < q.band.length; i++){
6     if (q1.qMin <= qMin + i <= q1.qMax) {
7      q.band[i] += q1.band[qMin + i - q1.qMin];
8     }
9     if (q2.qMin <= qMin + i <= q2.qMax) {
10     q.band[i] += q2.band[qMin + i - q2.qMin];
11    }
12   }
13        return q;
14 }
```

Example of Internal QBand Addition: The operands in internal QBand addition are any QoS properties. Here we still use response time as example. Let q1 is the response time QBand of a task t1, and three new web services for the task t1 come, the response times of the three services are q1.qMin+1, q1.qMin+2 and q1.qMin+3 individually, then the response time QBand of the three new web services is q2. The new QBand of the task t1 is q12. In contrast with the old QBand q1, q12.qMax becomes bigger because the new service with a RT value of [q1.qMin+2] joins in the candidate service set. The numbers of achieved QoS values for the task t1 update. For example, there are two candidate web services for t1 with a RT value of [q1.qMin+1]. With a new service of a RT value of [q1.qMin+1] joins in the candidate service set.

The numbers of achieving the RT value of [q1.qMin+1] increases from 2 to 3 shown in the new QBand q12 in Fig. 6.

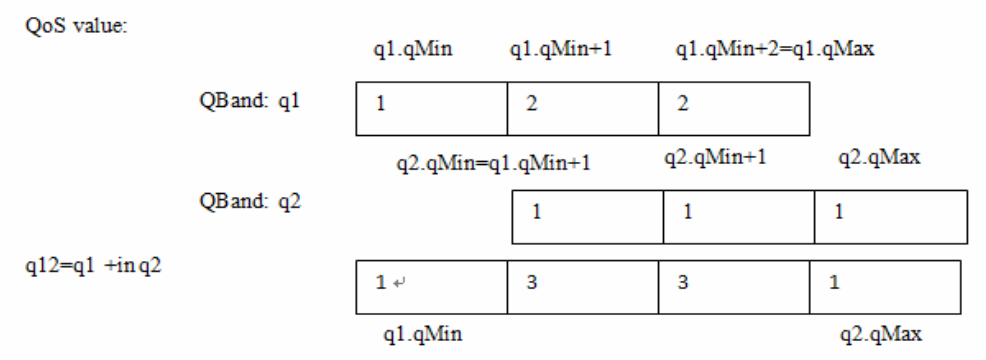


Fig. 6. Illustration of QBand internal addition

```
/*QBand –in is used to update the QBand of the task when candidate web ser-
vices is not provided.
  q1 is the QBand of the task, q2 is the QBand formed by the removed web
  services from the candidate service set of the task.
  Process:
   To update the QBand of the task,
     1) create a new QBand for the task (line2-4)
     2) subtract the number of the removed services for the task in the QoS
        boundary (line5-7)
     3) modify the QoS boundary (line8)*/
1 QBand –in (QBand q1, QBand q2){
2   integer qMin = q1.qMin;
3   integer qMax = q1.qMax;
4   QBand q = new QBand(qMin, qMax);
5   for (integer i = 0; i < q.band.length; i++) {
6           q.band[i] = q1.band[qMin+i-q1.qMin]- q2.band[qMin+i-q2.qMin];
7   }
8         Trim(q);// remove the zero elements in the two ends of q.band, and
              update q.qMin and q.qMax
9   return q;
10}
```

Example of Internal QBand Subtraction: The operands in internal QBand subtraction are any QoS properties. Here we still use response time as example. Let q1 is the response time QBand of a task t1, and two old web services for the task t1 close, the response times of the two services are q1.qMin+2 and q1.qMin+3 individually, then the response time QBand of the two old web services is q2. The new QBand of the task t1 is q12. With the old service of a RT value of [q1.qMin+3] close, the maximum RT in candidate services becomes q1.qMin+2. Thus the qMax of the new QBand q12 becomes q1.qMin+2. The function "Trim (q)" deals with this problem. The numbers of achieved QoS values for the task t1 update. For example, there are two candidate web services for t1 with a RT value of [q1.qMin+1]. With an old service of a RT value of [q1.qMin+2] close. The numbers of achieving the RT value of [q1.qMin+2] decreases from 2 to 1 shown in the new QBand q12 in Fig. 7.

The QBand q1 is the old QBand of a task t1, q2 is the QBand of closing services from the candidate service set, so q1 and q2 have the following relationships:

$$q1.qMin \le q2.qMin, q1.qMax \ge q2.qMax \quad \textbf{(1)}$$

$$\forall i \in [0, q2.bank.length] \rightarrow (q2.ban[i] \le q1.bank[q2.Min + i - q1.qMin] \quad \textbf{(2)}$$

We call q1 ≥ q2 when q1 and q2 have the relationships.

The QBand q12 is the QBand of left candidate services for the task t1. To make q12 valid, which means q12.band[i] ≥ 0, q1 should be greater than or equal to (≥) q2.

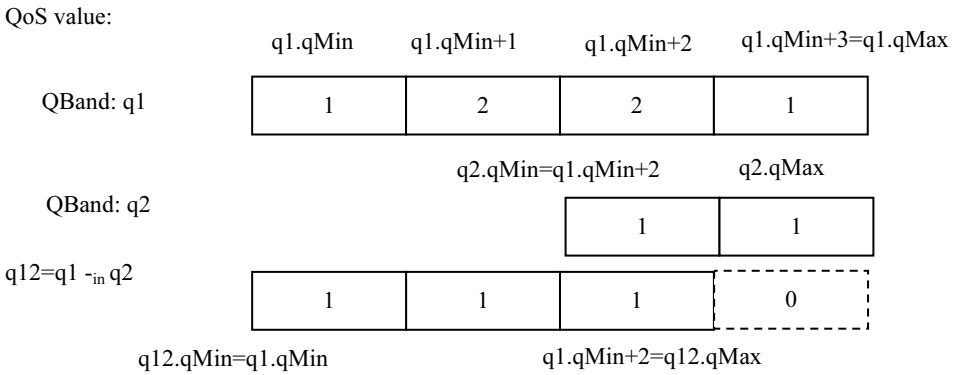


Fig. 7. Illustration of QBand internal subtraction

In the case that QoS of existing web services change, we use internal QBand subtraction and addition to update the QBand. We consider the changing services as closing web services with old QoS values and new web services with changing QoS values. For example, assuming the RT value of a web service changes from [q1.qMin+2] to [q1.qMin+1], then it's considered that a web service of RT value of [q1.qMin+2] closes and a web service of RT value of [q1.qMin+1] adds into the candidate service set.

We call QBand operations between tasks as QBand inter-operations such as QBand addition between tasks. It has the following relationship between QBand inter-operations and internal operations.

$$q \circ (q1 \pm_{in} q2) = (q \circ q1) \pm_{in} (q \circ q2) \quad \textbf{(3)}$$

where q1 and q2 are the QBand for the same task, $\circ$ is the QBand operators for the different tasks.

With the two internal QBand operations and Eq. (3), we can update the QBand of users' task incrementally instead of recalculation from the beginning. Let q1 is the QBand of a task t1 and q2 is the QBand of the task t2, $q12_{old}$ is the QBand of the composite task of t1 $\circ$ t2, $q12_{old} = q1 \circ q2$. $q2_{inc}$ is the incremental (positive or negative changing) QBand of the task t2 caused by changing candidate services for the task t2. Then $q12_{new}$ change $q12_{old}$ from with the increment $q1 \circ q2 \circ q2_{inc}$

$$q12_{new} = q1 \circ (q2 \pm_{in} q2_{inc}) = (q1 \circ q2) \pm_{in} (q1 \circ q2_{inc}) = q12_{old} \pm_{in} (q1 \circ q2_{inc}) \quad \textbf{(4)}$$

3.3 Appearance of QBand

The QBand, indicating the QoS implementation levels of web service composition solutions for a task, provides an unintuitive for human users in the array named band in the structure of QBand like [2 3 3…1] shown in Fig. 4. To represent the distribution of the service composition solutions visually, we develop a kind of grey-scale diagram to represent QBand. We call it QBand spectrogram. Different gray scales represent numbers of web service composition for the complex task. The higher a grey-scale value is, the more combinations of different candidate services in one QoS value are. The advantage of the spectrogram is to use the contrast between white and black to show the distribution and natural clusters of service compositions which can be perceived intuitively by human. An example of combination to fulfill a complex task and corresponding data structure is shown in Fig. 8. We design 5 web services with different characteristics of QoS used for candidate web services for each task. The amount of service composition of this complex task is about 1.95×10^6. The corresponding QBand for the complex task is shown in Fig. 9. As shown by the QBand spectrogram, 594ms is the q.qMin, the minimum RT of web service composition for the complex task and 1854ms is the q.qMax, the maximum RT. The bigger q.band[i] is, the brighter the corresponding of the figure is. Different gray scales represent RT levels of web service composition for the complex task. The higher a RT gray-scale value is, the more combinations of different candidate services in this value are. In this spectrogram of RT implementation levels, 909ms RT has maximum grayscale which means the most service composition of 909ms RT for the complex task.

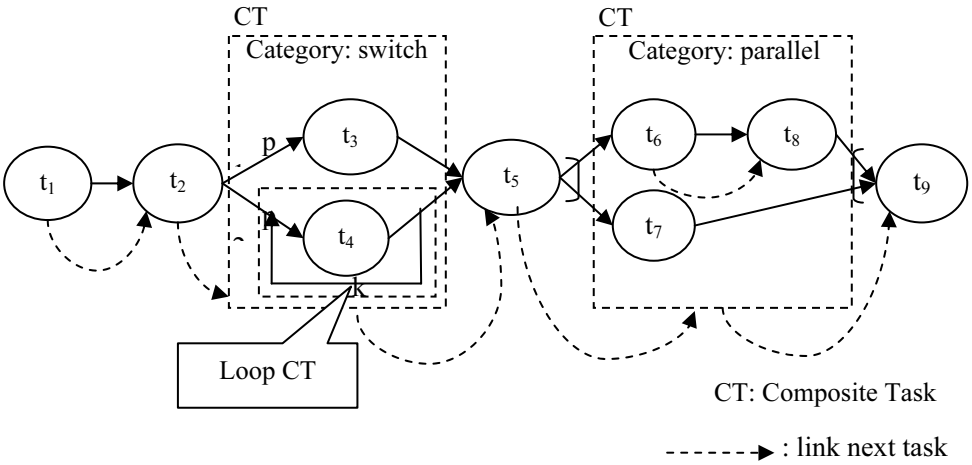


Fig. 8. An example of combination to fulfill a complex task and corresponding data structure

Fig. 9. QBand spectrogram for response time

The QBand spectrogram provides the user intuitive information about the current QoS implementation levels of web service composition for his complex task, and then the user can select the range according to their QoS requirements and current QoS implementation levels shown in Fig. 13 and Fig. 14. The spectrogram provides users with an overview of QoS implementation levels of web service composition, and helps users to determine QoS constraints of their composite services. The QBand spectrogram is just for one QoS property. Each QoS property has its own QBand spectrogram.

4 Illustration and Evaluation

4.1 Illustration of the calculation of QBand

We use a brief example task in Fig. 10 to illustrate how QBand calculates the QoS implementation levels of web service composition for the user's task, including the QoS boundary of the service composition solutions and the numbers of the service compositions available for each value in QoS boundary. We compare QBand with the exhaustive method, which is the primary and only method to fulfil the same function currently to show why QBand is better than the exhaustive method to fulfil the same function.

The simple example task contains 3 sequential sub-tasks t_1, t_2 and t_3. The subtask t_1 has 3 candidate web services with the same QoS property response time of 10ms. QBand (t_1) is shown in Fig. 10. It contains the meaning that t_1 has 3 candidate web services with RT of 10ms. t_2 also has 3 candidate web services. Two of them have RT of 10ms. The other one has RT of 11ms. QBand (t_2) is shown in Fig. 10. t_3 has 2 candidate web services whose RT is 9ms. To get the RT values of the composite task of $t_1+t_2+t_3$ and the numbers of service composition for each RT value, i.e. get the RT implementation levels of web service composition for the task, first we get the RT implementation levels of web service composition for the task t_1+t_2, then we get the QBand ($t_1+t_2+t_3$). According to combinational principle, the amount of service composition of the composite task t_1+t_2 is 9, and the RT values are 20 and 21ms. The service composition of 20ms RT can be produced by $ws_1,1$ and $ws_2,1$, or $ws_1,1$ and $ws_2,2$. The amount of service composition of 20ms RT is 6. The service composition of 21ms RT can be pro duced by $ws_1,1$ and $ws_2,3$, or $ws_1,2$ and $ws_2,3$. The amount of service composition of 21ms RT is 3. The exhaustive method lists all service composition and counts the corresponding numbers of service composition for each RT value number. The calculation number of the exhaustive method is 3*3=9. 3 is the number of candidate web services for t_1, and the other 3 is the number of candidate web services for t_2.

On the contrast, to get the RT implementation levels of web service composition for the task t_1+t_2, QBand knows there are 3 candidate services of 10ms RT for t_1 and there are 2 candidate services of 10ms RT and 1 candidate services of 11ms RT for t_2. Based on these information, QBand method figures out the RT values of service composition of the composite task t_1+t_2 are 20ms and 21ms. The amount of service composition of 20ms RT is 3*2=6. The amount of service composition of 21ms RT is 3*1=3. The calculation number of the QBand method is 1*2=2. 1 is the length of QBand (t_1), and 2 is the length of QBand (t_2). In the same way, we can get the RT implementation levels of web service composition for the task $t_1+t_2+t_3$, the calculation number of the exhaustive method is 3*3*2=18 while that of QBand method is 1*2*1=2. By comparing the calculation number of the two methods, we can see the QBand method has better performance to get QoS implementation levels compared with the exhaustive method intuitively.

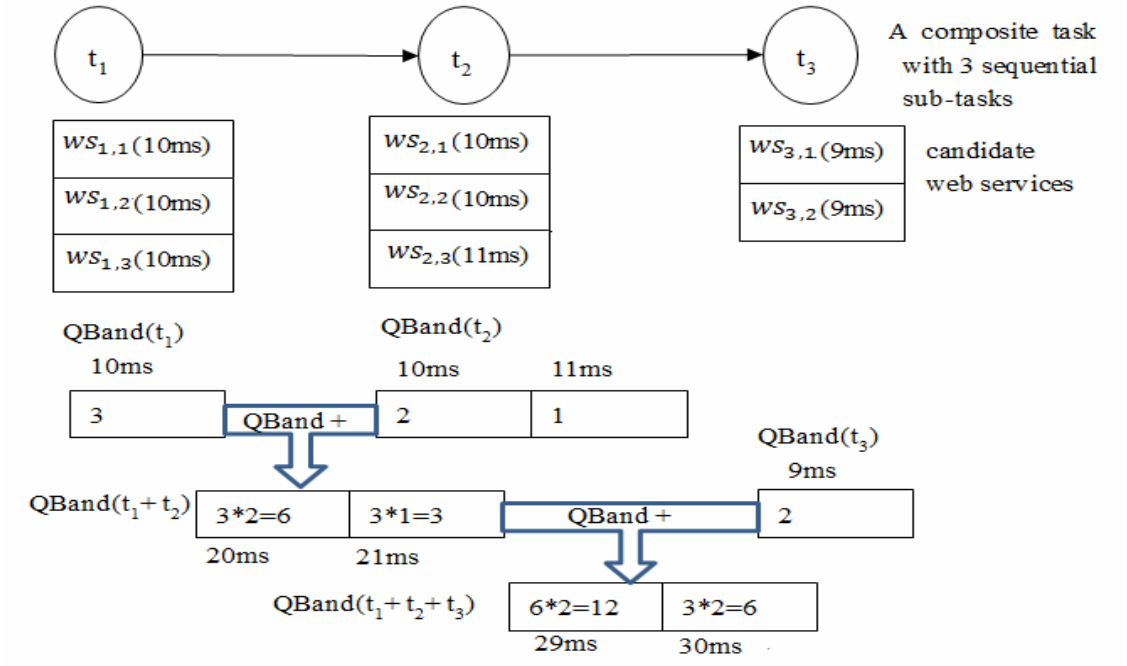


Fig. 10. Illustration of exhaustive method and QBand method to form QoS implementation levels of service compositions

4.2 Evaluation of QBand

In the above statement, we use a simple example task in Fig. 10 to illustrate how QBand works. The example task just has sequence logic, one of four execution logic for the brief of illustration. To validate our approach, we carried out experiments using a complex task in Fig. 8 which contains all the four basic logic aforementioned. We compared our approach with the exhaustive method to calculate the QoS implementation levels of web service composition solutions for the complex task. First, 5 candidate web services are assigned for each task. The QoS of these web services are randomly generated by Java.Math.random() in each experiment. The experiment number is 10. It is shown that QBand has better and stable performance than the exhaustive method in Fig. 11. The average time that QBand consumed is 2.15ms while that of the exhaustive method is 317ms. The performance difference between them is obvious.

We compared the results of QoS values and the corresponding service composition solutions calculated by the exhaustive method and QBand method, they are same. This makes sure the correctness of our approach (See Table 2). Since the data obtained experimentally is large, this paper only picks out 30 data for demonstration. Table 2 is the comparison of values of response time and the number of the corresponding service composition solutions calculated by the exhaustive method and QBand method. The results are consistent by comparison.

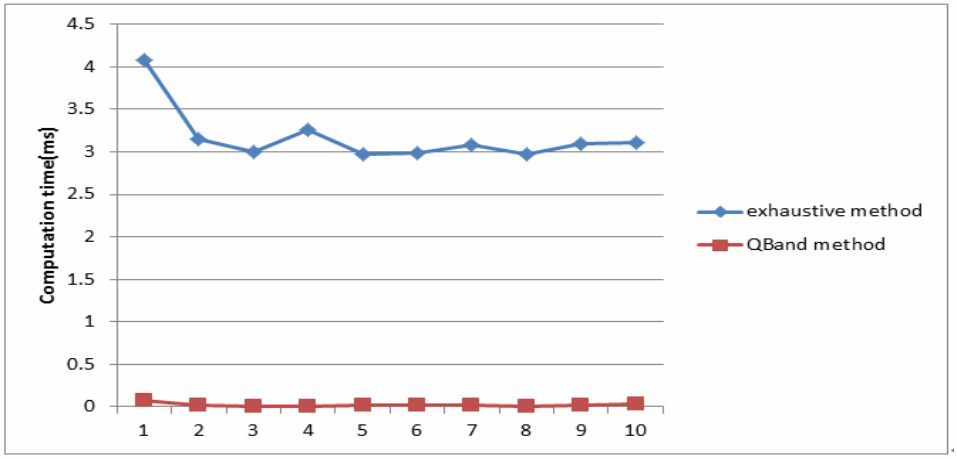


Fig. 11. Comparison of time consumed between exhaustive method and QBand method to form current QoS implementation levels of service composition solutions

Table 2. Comparison of the web service composition solutions calculated by the exhaustive method and QBand method (partial, 30 in this table)

response time (ms)	QBand	exhaustive method	response time (ms)	QBand	exhaustive method	response time (ms)	QBand	exhaustive method
601	445	445	611	716	716	621	852	852
602	441	441	612	621	621	622	998	998
603	538	538	613	654	654	623	942	942
604	464	464	614	734	734	624	992	992
605	497	497	615	779	779	625	1009	1009
606	526	526	616	743	743	626	1103	1103
607	613	613	617	707	707	627	1061	1061
608	560	560	618	896	896	628	1132	1132
609	568	568	619	844	844	629	1133	1133
610	593	593	620	865	865	630	1225	1225

We increased the number of candidate web services by the step size of 5 for further validation. The experiment number is still 10 in every increment of the step. The results are shown in Table 3. When the number of candidate web services is 10, the amount of service compositions for this complex task is 10^9. The number of service compositions increased by 512 times, compared with the former. The average time that QBand consumed in the experiments doesn't increase quickly while that of the exhaustive method increases by two orders of magnitude from 3.17×10^2 to 5.19×10^4. The performance difference between them further increases. When the number of candidate web services becomes 15, the amount of service compositions increases by about 38 times again, and the average time taken by the exhaustive method becomes too long to be acceptable. Although the time of QBand also increases, it increases slowly. It is acceptable compared to the time of the exhaustive method. During the experiments, we compared the RT implementation levels produced by the exhaustive method and QBand method, and all the results are the same, ensuring the correctness of our approach.

Table 3. Comparison of the average time consumed between the exhaustive method and QBand method in the cases of different candidate web services. ws is short for web services. ms=milliseconds

Number of ws	5	10	15
T1(ms)=Time(exhaustive method)	3.17×10^2	5.19×10^4	2.13×10^6
T2(ms)=Time(QBand method)	2.15	3.8	17.58
T2/T1	1.47×10^2	1.37×10^4	1.21×10^5

5 Prototype Implementation

We developed a prototype implementation to illustrate the QoS-based service composition and the scenario that the users use QBand to set QoS constraints. The dataset used in our case study is QWS [9], an open QoS dataset for web service research. We still use the complex task in Fig. 8 as example to show the process of using our approach for QoS-based service composition. The initial interface for defining QoS constraints is shown in Fig. 12. It shows the functions of the system, including the setting of QoS ranges and preferences and the display of QoS constraints that users set.

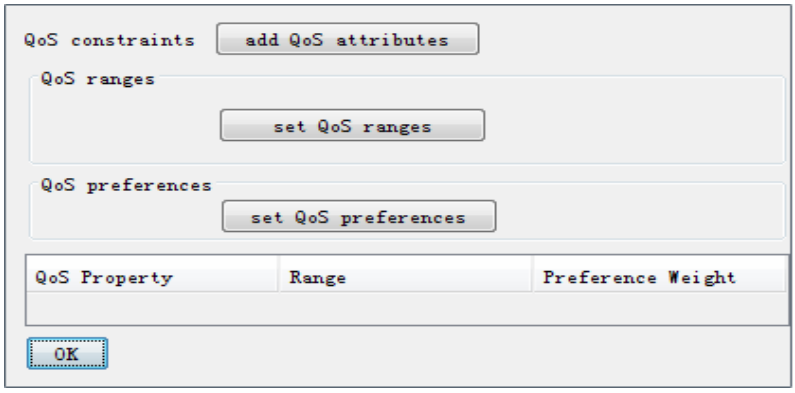


Fig. 12. Initial interface for defining QoS constraints

A user has demand on the ranges QoS properties of service composition solutions for his/her task. He /She wants QoS properties to reach the degrees such as high, very high and so on among all possible composition solutions. But the user doesn't know that range. Then QBand provides the user the current QoS implementation levels of web service composition for his complex task. The user can determine the range of QoS properties with the QBand. In Fig. 14, when the cursor lingers over a part of the QBand, the corresponding value of the response time to the part displays. Here the user set the range of Response Time (874, 969), in which there are more service compositions.

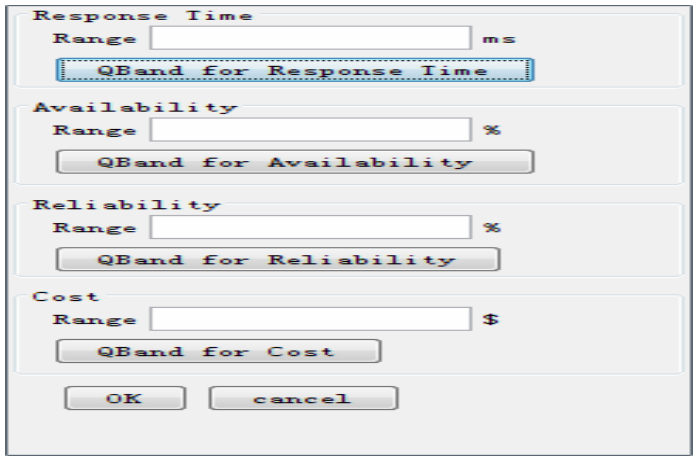


Fig. 13. Specifying QoS ranges

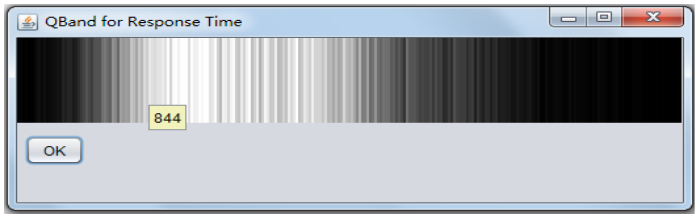


Fig. 14. QBand spectrogram for response time

The prototype implementation provides a preference expression to help users express their preferences on QoS attributes of web service based on consumer preference axioms [10]. A user arranges QoS attributes in preference expression to express his preference order and uses linguistic variables to express the degree of his preferences, thus forming the complete information of the user's preferences on QoS attributes. Then AHP [11] is used to convert the semantic description of the preference in the expression into the preference weights of QoS attributes. In contrast with the diffculty of setting preference weights directly, it is easy and intuitive for the user to put QoS properties in preference order and specify preference degree word between two QoS properties. In Fig. 15, the user expresses RT has highly important priority over Availability and Cost. More details are explained in our work [12-13].

Fig. 15. Expressing the preference with the preference expression

If the above QoS settings don't satisfy the user's QoS requirements, the user can set the exact numerical range and preference weights. In Fig. 16, the user set Availability (50,100) manually.

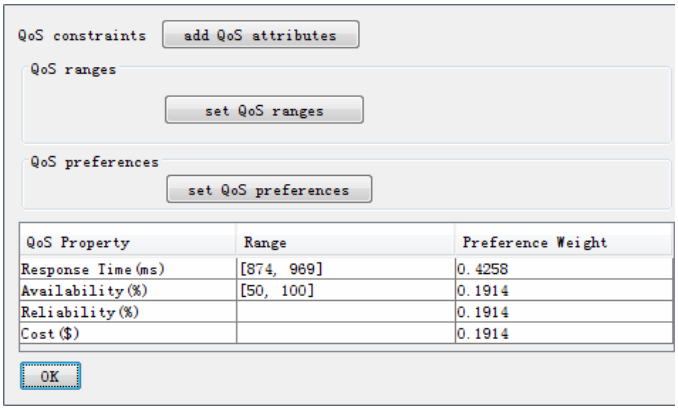


Fig. 16. Specifying QoS constraints manually

The result is shown in Fig. 17. It shows the user's QoS constraints (QoS ranges and preference weights) and the optimization solution of the web service composition.

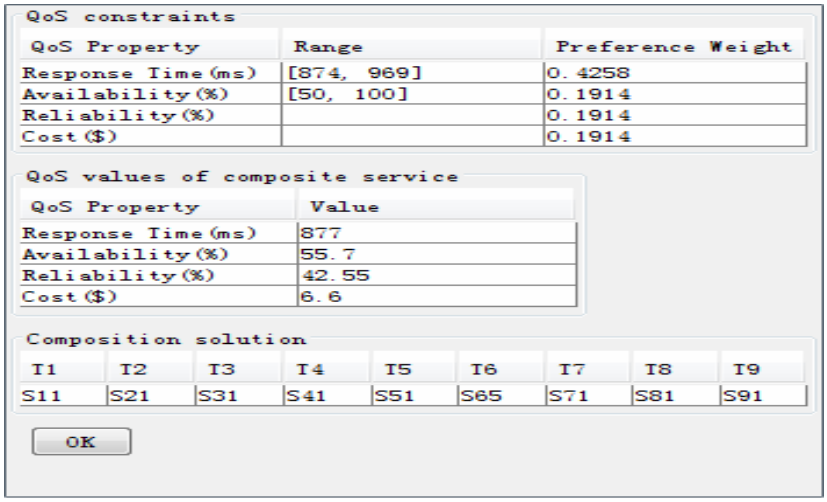


Fig. 17. Result of composition solution

If the solution doesn't meet the user's requirement after the user experience the composite web services, the user can adjust the QoS constraints based on the QoS of the composition solution and the former QoS constraints.

6 Related Work

With more and more web services published on the Internet, multiple functional similar services could appear. Then QoS becomes an important aspect in service selection. And QoS-based web service composition becomes a hot research topic in service composition. There are some major aspects related or involved in QoS-based service composition, including QoS model, QoS collection and composition algorithm. As an indicator of distinguish similar services, QoS model is proposed in [14-17], including response time, reliability, availability, cost and domain properties, etc. QoS data are constantly monitored and collected by web service search engines like seekda and WSCE [18] to support QoS-based service composition. QoS-based service composition is modelled as constraint satisfied problem. Different approaches have proposed such as IP (Integer Programming) [19-20], GA [21-23], PSO (Particle Swarm Optimization) [24-26] and etc. More details of previous approaches on QoS-based web service composition can be found in the surveys [2-3, 6]. Most of these approaches focus on implementing service composition algorithms itself. They just provide primary ways to input QoS constraints assuming users can provide their QoS constraints accurately. On the contrary, users usually don't define their QoS constraints accurately because of their non-clarity of QoS requirements. More comprehensive ways are desirable, to help users to complete QoS constraints implicitly or explicitly.

Fuzzy theory has been introduced to handle vague description of QoS partially [27-29]. In these approaches, QoS are categorized into several groups, with linguistic variables such as "poor, medium and high" to represent these groups. Then web services are assessed based on those QoS values. Fuzzy and exact QoS requirements are both supported in [30]. The paper considers the inexperience end users in the QoS-based web service selection system. The paper proposes relaxation orders to get the services partially matched with QoS requirements. The advantage of applying fuzzy theory into service selection is to assist users to determine vague QoS constraints by using linguistic variables and to deal with subjective and vague assessments in service selection according to fuzzy logic. The main issue of these approaches is that they require considerable and expert efforts to determine the corresponding boundaries of linguistic variables and membership functions reasonably. What's the range of the QoS levels such as "poor, medium and high" of current service composition for a task? It needs some tools to provide these information to reduce the efforts.

In addition, setting QoS constraints not only depend on users' QoS requirements expectation, but also depend on current QoS of web services provided on the service market. Users want more information about the condition of web service on the Internet. They want more information on current QoS implementation levels of service compositions about their tasks, such as the distribution of service compositions in different QoS levels. To address these issues, it needs the calculation of QoS and its corresponding number of service compositions implementing the users' tasks. However, there is the exhaustive method that is the only and primary method to fulfil the function currently. The exhaustive method takes much time. Therefore, we propose a novel data structure called QBand better than the exhaustive method to help users specify their QoS requirements expressively and flexibly.

7 Conclusions and Future Work

This paper proposes a data structure QBand to help users to set their QoS constraints by providing more information about current QoS implementation level of web services implementing the task. QBand operations in QBand space are defined to calculate QBand of composite tasks and update QBand incrementally. QBand is mainly used for quantitive attributes. For qualitative attributes, QBand can be applied to them by quantization. For example, the reputation of web services can be quantitated by rating. QBand not only is used for indicating the QoS implementation levels of web services, but also for indicating quantitive attributes of other system components. QBand assists users to set QoS ranges according to current QoS level. With QBand, users can set their QoS constraints according to their QoS requirements expectations, combined with current QoS implementation levels of service compositions available on the

Internet. The algorithm can also be used for indicating quantitive attributes of other composite system, such as the products from the assembly line by slight modification.

At present, we just finish the algorithm design of QBand. There is still work for improvement in future. QBand is defined according to several general QoS attributes. It might be slightly modified and extended according to the characteristics of the new and specific QoS attributes. Integration of QoS collection technologies like seekda.com to provide the QoS data of candidate web services is our next-step main work to make the QBand into practice.

Acknowledgement

This paper is supported by the National Natural Science Foundation of China under Grant No. 61170087, 61133010 and 61520106006.

References

[1] Z. Liangzhao, B. Benatallah, A.H.H. Ngu, M. Dumas, J. Kalagnanam. H. Chang, QoS-aware middleware for web services composition, IEEE Transactions on Software Engineering 30(5)(2004) 311-327.

[2] A. Strunk, QoS-aware service composition: a survey, in: Proc. of the 8th European Conference on Web Services(ECOWS), 2010.

[3] B. AL-Shargabi, A. Sabri, A.E.L. Sheikh, Web service composition survey: state of the art review, Recent Patents on Computer Science 3(2)(2010) 91-107.

[4] G. Canfora, M. Di Penta, R. Esposito, M.L. Villani, An approach for QoS-aware service composition based on genetic algorithms, in: Proc. of the 2005 Conference on Genetic and Evolutionary Computation, 2005.

[5] D. Schuller, A. Polyvyanyy, L. Garćıa-Bãnuelos, S. Schulte, Optimization of complex QoS-aware service compositions, in: Proc. of the 9th International Conference on Service Oriented Computing, 2011.

[6] Y. Shi, X. Chen, A survey on QoS-aware web service composition, in: Proc. of International Conference on Multimedia Information Networking and Security, 2011.

[7] M. Christel, K.C. Kang, Issues in Requirements Elicitation, Technical Report CMU/SEI-92-TR-012, CMU / SEI, 1992.

[8] J. Cardoso, Quality of service and semantic composition of workflows, [dissertation] University of Georgia, 2002.

[9] E. AL-Masri, Q.H. Mahmoud, QoS-based discovery and ranking of web services, in: Proc. of the 16th International Conference on Computer Communications and Networks, 2007.

[10] H.R. Varian, Intermediate Microeconomics: a modern approach, 7th ed., Truth & Wisdom Press, Shanghai, 2009.

[11] T.L. Saaty, A scaling method for priorities in hierarchical structures, Journal of Mathematical Psychology 15(3)(1977) 34-281.

[12] G. Wang, L. Zhang, K. Nie, Multi-strategic approach of fast composition of web services, in: Proc. of 14th Asia-Pacific Web Conference, 2012.

[13] G. Wang, L. Zhang, J. Jiang, W. Jiang, QoS-based service composition under various QoS requirements, in Proc. of 20th Asia-Pacific Software Engineering Conference, 2013.

[14] L. O'Brien, L. Bass, P. Merson, Quality attributes and service-oriented architectures, CMU/SEI, 2005.

[15] K. Kritikos, B. Pernici, P. Plebani, M. Carro, A survey on service quality description, Computing Surveys 46(1)(2013) 1-58.

[16] QoS for Web Services: Requirements and Possible Approaches. <http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/>.

[17] S.H. Zhao, G.-X. Wu, S.-F. Zhang, Q. Fang, K. Yu, Review on SOA of quality of service research, Computer Science

36(4)(2009) 16-21.

[18] E. AL-Masri, Q.H. Mahmoud, WSCE: A crawler engine for large scale discovery of web services, in: Proc. of International Conference on Web Services, 2007.

[19] A.F. Huang, C.W. Lan, S.J. Yang, An optimal QoS-based web service selection scheme, Information Sciences 179(19)(2009) 3309-3322.

[20] J. Li, Y. Zhao, H. Sun, Z. Zheng, D. Ma, DH4SS: a distributed heuristic for QoS-based service selection, Int. J. Web and Grid Services 7(4)(2011) 388-409.

[21] G. Canfora, M. Di Penta, A lightweight approach for QoS aware service composition, in Proc. of International Conference on Service Oriented Computing, 2004.

[22] Z. Jiang, J. Han, Z. Wang, An optimization model for dynamic QoS-aware web services selection and composition, Chinese Journal of Computers 32(5)(2009) 1014-1025.

[23] A. Klein, F. Ishikawa, S. Honiden, Effcient heuristic approach with improved time complexity for QoS-aware service composition, in: Proc. of International Conference on Web Services, 2011.

[24] X. Fan, C. Jiang, X. Fang, Z. Ding, Dynamic web service selection based on discrete particle swarm optimization, Journal of Computer Research and Development 47(1)(2010) 147-156.

[25] H.-L. Cai, L. Lu, F. Kun, W. Li, Web services recommendation based on bpso, Journal of Shenzhen University Science and Engineering 27(1)(2010) 49-55.

[26] Z. Liang, H. Zou, F. Yang, R. Lin, A hybrid approach for the multi-constraint web service selection problem in web service composition, Journal of Information & Computational Science 9(3)(2012) 3771-3781.

[27] Z. Li, F.C. Yang, S. Su, Fuzzy multi-attribute decision making-based algorithm for semantic web service composition, Journal of Software 20(3)(2009) 583-596.

[28] P. Xiong, Y. Fan, QoS-aware web service selection by a synthetic weight, in: Proc. of International Conference on Fuzzy Systems and Knowledge Discovery, 2007.

[29] I. Sora, D. Todinca, Dealing with fuzzy QoS properties in service composition, in: Proceeding of 10th IEEE International Symposium on Applied Computational Intelligence and Informatics, 2015.

[30] D. Mobedpour, C. Ding, User-centered design of a QoS-based web service selection system, Service Oriented Computing and Applications7(2)(2013) 117-12.