# Multi-Path QoS-Aware Service Composition

**Osama Kayed Qtaish*, Zulikha Bt Jamaludin**, Massudi Mahmuddin*****
*(School of Computing, Universiti Utara Malaysia
** (School of Computing, Universiti Utara Malaysia
*** (School of Computing, Universiti Utara Malaysia

## Abstract
**The goal of QoS-aware service composition is to generate optimal composite services that satisfy QoS requirements defined by clients. However, when compositions contain multiple execution paths, it is difficult to generate a solution that simultaneously optimizes all the execution paths involved in the composition at the same time, while meets global QoS constraints imposed by clients. Furthermore, finding an exact optimal solution by evaluating all possible combination, results in a combinatorial problem which is known to be NP-hard. This paper aims to tackle these major issues by proposing an innovative approach that suits for multi-paths compositions. In this approach, an optimization mechanism is proposed that computes the optimization by considering only the path that will be followed during the execution. To do so, a data mining method is adapted to predict, at runtime and just before the actual execution, the path that will be followed during the execution. By optimizing only the predicted path, it is expected to generate an optimal solution that delivering the best possible QoS performances, while guarantee meeting global QoS constraints. For solving the optimization problem, the problem is modelled as multi-dimensional multi-choice knapsack problem (MMKP). Then, heuristic algorithms are applied to solve it. By applying heuristic, it is expected to significantly reduce the computation efforts.**

**Keywords –** data mining, heuristic algorithm, QoS, service selection, web service

## 1. INTRODUCTION
Recently, Service Oriented Computing (SOC) has gained a considerable momentum from both industry and academia as a new emerge paradigm to develop rapid, low cost, and loosely coupled software systems. This vision is captured by Service Oriented Architecture (SOA) design principles such as loosely coupling. SOA is "*a way of designing a system so that it can provide services to end users and/or other applications in the network*" [1]. Web services are the leading technology for implementing SOA.

Web services are software systems designed to perform functionality. They are loosely coupled, network available, platform-independent, self-described interface, and can communicate using standard protocols. They are published by service providers (i.e., organizations that provide service descriptions and ensure service implementations), located, and invoked by clients (organizations). Clients in turn, can utilize web services without the need to install it [2].

One of the main benefits gained from implementing web services and SOA is the ability to compose new functionality out of existing outsourced web services into so-called composite services. The process of creating such composite services is called web service composition [3].

SOA along with service composition technology have changed the way software engineers design and develop business processes. Rather than developing entirely new processes; SOA processes are developed by composing network available web services [4]. Business process is a set of related tasks or activities that designed to realize a specific organizational goal. Each task (also referred as an abstract web service) of such a business process can be accomplished by a single outsourced web service hosted by external partners. This vision enables agile collaborations between several business partners, and thus decreases the cost of building business processes.

Under this scenario, a software engineer, during the design time, defines the business process by identifying and arranging the abstract services/tasks (a semantic description of a specific functionality, e.g. invoking a credit card). Based on the semantic descriptions of the abstract services, many functionality equivalents web services (services that perform similar functionality also called candidates) can be discovered for each abstract web service. Then, the service selection can be performed dynamically at runtime by selecting the best outsourced services that can accomplish the abstract services' functionality [5]. One of the most substantial selection factors that can be served as selection criteria between those equivalent services is the Quality of Service (QoS) criteria. QoS represents web service's non-functional characteristics such as cost, response time, availability, reliability.

Having QoS characteristics as selection criteria; the goal of QoS-aware service composition process is to select one candidate web service for replacing each abstract web service such that the entire QoS of the business process (hereafter used interchangeably with terms "composite service" and "composition") is optimized while clients (i.e., organizations) QoS requirements are satisfied. These requirements include QoS global constraints and preferences. QoS global constraints are constraints imposed by the clients on the whole composition. For example, a client could specify that the total cost of the composite service execution must be less than 2000 Dollars, at the same time; they could prefer the composite service with high security and/or low response time.

On the other hand, compositions are operated in highly dynamic environments. In such environments, different possible scenarios may occur at runtime, making the real time compositions facing unanticipated changes. In the case that the composition is defined at design time, it is desirable to support the expectations that can be anticipated by composition engineers. *Flexibility by configuration* or *flexibility by design* referred to the structural properties of a composition which allow it to flexibly respond to different scenarios anticipated by composition engineer at design time [6]. Flexibility by configuration by conditional structures (include OR split/join, XOR split/join and the OR split with m-out-of-k join patterns) provides the ability to incorporate multiple execution paths within the composition definitions at design time [7]. The result is distinct set of multiple execution paths. Each path represents a scenario that can be followed during the execution of a composition instance [6].

Within the context of multiple execution paths, optimization algorithms are required to compute the optimization by considering all the execution paths involved in a composition. However, it is difficult to generate a solution that simultaneously optimizes all execution paths involved in the composition at the same time, and satisfy global QoS constraints imposed by clients.

For service optimization, finding an exact optimal solution required a strategy based on evaluating all the possible combinations to find the optimal one. Such a straightforward strategy results in a combinatorial problem, and the computational complexity to find a solution for this problem is non-deterministic polynomial time (i.e., NP-hard) [8]. It is impractical and time consuming to evaluate all these combinations to find the optimal one. Therefore, solutions based on heuristic methods, although they deliver near-to-optimal solutions, are desired [9].

The aim of this work is to tackle these major issues occurs when optimizing compositions including multiple execution paths. For this purpose, an innovative QoS-aware service composition approach is proposed that efficiently suits for multi-path compositions. In this approach, an optimization mechanism is proposed based on the combination between a data mining method and heuristic algorithms. This mechanism allows computing the optimization by considering only the path that will potentially be followed during the execution of a business process. By using our approach: (i) it is expected to always generate solutions with the best QoS possible, at the same time, meet global QoS constraints, (ii) it is expected to significantly reduce the computation efforts for finding optimal solutions**.**

## 2. RELATED WORKS

Different techniques are proposed to handle the optimization of multi-path compositions. For example, Yu et al. [10], Canfora et al. [11], Wang et al. [12], and Lecue [13] compute the optimization assuming that a certain path will be more likely executed than another according to probability of paths execution. The assumptions are based on stochastic information indicating the probability of paths being executed at runtime. Paths' probability of executions can be estimated either by inspecting the system logs which contain information about the past executions or can be specified by the composition engineers. However, the results of such assumptions may be false; consequently, the generated solutions may not have the best QoS performance.

Zeng et al. [14], [15] optimize each path separately by decomposing the composition into execution paths, and after the optimization process, the execution paths are aggregated into an overall composition that consists of all paths. If there is a common abstract service that belongs to more than one path, the system identifies the hot path for the considered web service. They define the hot path as the path that has been most frequently used to execute the considered service. However, in the case that the actual execution of the composition is not following the hot path, the executed path may not have the best QoS performance. Even worst, the executed path may not meet global QoS constraints.

Uker et al. [6] survey the QoS-aware service composition approaches in order to analyse the ability of the optimization algorithms to simultaneously generate optimal plans for all executions paths involved in compositions. They conclude that, within the context of multi-path compositions, it is difficult to generate a solution that simultaneously optimizes all the execution paths in the composition at the same time. In a subsequent paper [16], the authors address this problem by presenting an approach that enables users to bias the optimizations using a set of meta-metrics. These meta-metrics include: execution probability of an activity, previous execution history of each activity, and probability of occurrence. The approach aims to find an approximation solution for each path involved in the composition. A trade-off between the paths is made, which chooses a path to favor by using a set of meta-metrics. For each path, the meta-metrics are computed as the weighted average of the aggregate values of meta-metrics. Then, the selection problem is solved using integer programming (IP) solution. Similar to this approach, Neelavathi et al. [17] also propose to use meta-metrics to resolve the conflicts caused by optimizing multiple execution paths. Their meta-metrics represent a priority of execution activities in execution paths. However, in these approaches, service selection optimization is biased using a set of meta-metrics. These meta-metrics are based on assumptions either assigned by the composition developers or estimated from the log trace records. These assumptions may be false. Consequently, the solutions obtained from these approaches may prove to be suboptimal for some execution paths. Even worst, it may violate the global QoS constraints.

In contrast to the above mentioned approaches, the solutions generated from our approach are expected to deliver the best possible QoS, at the same time, satisfy global constraints. This is because of the proposed optimization mechanism that designed to optimize only the path that will be followed during the execution of a composition.

## 3. THE PROBLEM MODEL

The problem model is formulated as follows:

- A set of tasks or abstract web services $AS = \{as_1, as_2, ..., as_a\}$, where $i = 1,..., a$, and $a$

represents the total number of abstract services involved in a composition.

- A set of service classes $S = \{S_1, S_2, ..., S_a\}$. For each class $S_i$, there is a set of functionality equivalent candidate service (also called concrete service) $S_i = \{s_1, s_2, ..., s_{b_i}\}$, that can execute the abstract service $as_i$, where $j = 1,..., b_i$, and the variable $b_i$ represents the number of candidate founds for abstract service $as_i$.

- More than one QoS characteristics values can be assigned for each candidate. Therefore, a QoS vector $q_{ij}$ is assigned for each candidate $s_j$. The vector contains the different QoS values represented by the index $k$, $q_{ij} = [\overset{1}{q_{ij}}, ..., \overset{n}{q_{ij}}]$, where $k = 1,..., n$.

- A vector of global QoS constraints imposed by clients $GS = [\overset{1}{GS}, ..., \overset{n}{GS}]$, where $k = 1,..., n$. It can contain none, one or $n$ global QoS constraints.

- A valid solution, i.e., a composition plan, can be obtained by assigning candidate services $s_j$ to each abstract service $as_i$ such that $s_j \in S_i$ for which the aggregated QoS values meets the given global QoS constraints, and the overall QoS value of the composite service is maximized.

Table 1 represents the notations used for the problem model.

**Table 1: Problem model notations**

| Notation | Meaning |
|---|---|
| Abstract service $as_i$ | Describing the desired functionality. |
| Service class $S_i$ | Is a collection of candidate services with a common functionality but different QoS characteristics value. |
| Candidate service $s_j$ | Candidate service $j$ from service class $S_i$ for abstract service $as_{i.}$ |
| QoS vector $q_{ij} = [\overset{1}{q_{ij}}, ..., \overset{n}{q_{ij}}]$ | QoS vector represents the different QoS values of candidate service $s_{ij}$ |
| QoS global constraints $GS = [\overset{1}{GS}, ..., \overset{n}{GS}]$ | Is a vector of all QoS global constrains imposed by the client on the whole composition. |

## 3.1 Modelling the structures

To define a composition, different structures (such as sequential, loop, etc.) can be used to connect the services. In this work, we focus on compositions that defined using sequential and conditional structures. Other structures, such as loop for example, may be reduced to sequential as in [15].

***Definition* 1: *Execution path (EP)*:** If a composition contains conditional structures, it has multiple sequential execution paths. Each execution path $EP_i$ represents a sequence of services $\{S_1, ..., S_i, ... S_a\}$. Each $EP_i$ takes only one branch in each conditional branching. For example, there are 3 execution paths in fig 1:

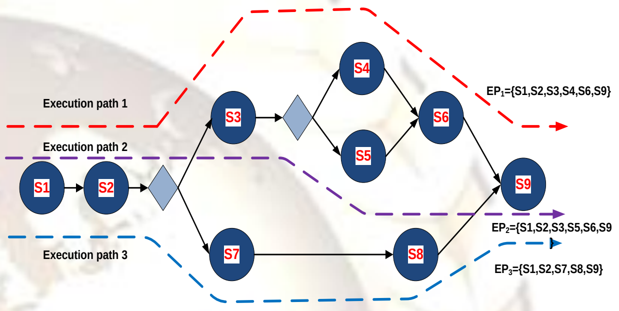

**Figure 1: Multiple execution paths composition**

$$EP_1 = \{S_1, S_2, S_3, S_4, S_6, S_9\}$$

$$EP_2 = \{S_1, S_2, S_3, S_5, S_6, S_9\}$$

$$EP_3 = \{S_1, S_2, S_7, S_8, S_9\}$$

and $EP_1, EP_2, EP_3 \subset AS$

***Definition 2: Predicted path*** **(*$EP_{pred}$*)**: is the execution path that will be followed during the execution of a composition. It can be one of the sequential execution paths defined above. $EP_{pred}$ is identified using the data mining method explained later.

## 4. MULTI-PATH QOS-AWARE SERVICE COMPOSITION

As illustrated in Fig. 2, QoS-aware composition process has four inputs: abstract composition (i.e., a set of abstract services connected using composition structures), a list of outsourced candidate web services discovered for each abstract service and their QoS characteristic values, and finally a client's (organization's) global QoS requirements. The desired output is an optimal composition plan.

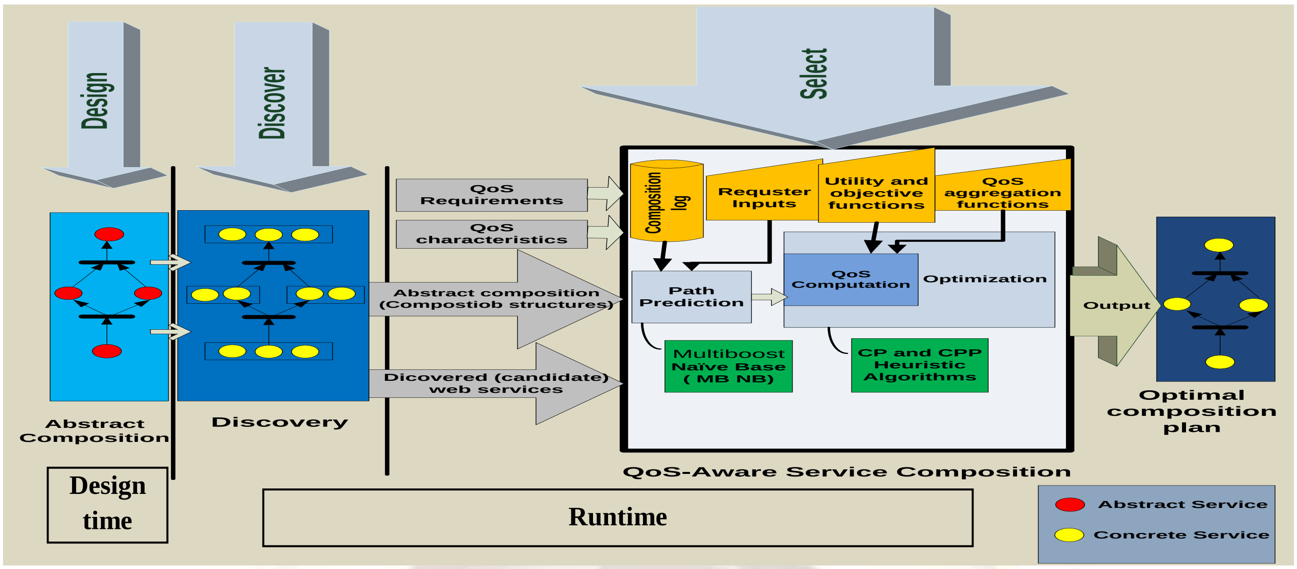

Figure 2: the proposed approach

In this work, the QoS-aware compositions process is carried out dynamically on instance by instance basis. It is started by predicting, at runtime and just before the actual execution of compositions, the path that potentially will be followed during the execution of a composition. To do so, the Multiboost Naïve Base (MB NB) machine learning algorithm will be applied on the composition log to learn how to classify the unknown classes. Note that this step will be carried out offline, so it does not affect the performance of the proposed approach. After training the algorithm, it will be ready to predict the path based on the data provided by the composite service's requester. At runtime, a client (i.e., a service requester) is required to provide data including personal data and data describing the condition of the service being requested. Then, the data needed for prediction are collected and formatted, and then input to a classifier to classify into target classes, i.e., composition paths. Based on the predicted path, CP and CPP heuristic algorithms will compute the optimization by considering only the predicted path.

## 4.1 Selection criteria

In our approach, we considered the following QoS characteristics as selection criteria: cost, response time, reliability, availability, security (encryption level), throughput, reputation and composability. The suggested QoS characteristics are identified based on our previous work [18], where we investigating and analyzing 25 QoS characteristics that were used in the area of web service and SOA.

Cost represents the amount of money that a service requester has to pay for a service provider for using its service. Response time is a typical measure of performance. It represents the total time required to complete a service request, which can be defined by the sum of the time a service need to process a request on the provider side (processing time), and the time needed to send a request and receive a response over a network [19]. For reliability, it represents degree that a service is able to correctly respond to a request in a specified time interval. The number of service's failures in minutes, days or months describes its reliability [20]. The availability of a web service represents the probability that the service is ready for access when required to immediate use [15], [20]. Security characteristic can include numerous aspects. It means providing confidentiality, authentication, authorization, encrypting data and non-repudiation. These security aspects can be provided in different level of policy by service providers [19], [20]. The only aspect that can be described with numerical value is the encryption level, which is a measurement that describes the length for a key that is used for encryption [21]. For throughput, it is a measure of service's productivity. It can be defined as the number of requests that the service provider can process in a given time period [19], [20]. Reputation represents a ranking that is provided by the users of a service based on their experience of using it. Reputation measures the trustworthiness of a service [15]. Finally, composability represents the probability that the service is executed as a member of the composition service [22].

## 4.2 QoS computation for web service composition

The QoS value of a composite service is computed from its constituent web services. The QoS value of a composite service SN is defined by the vector Q. This tuple contains the aggregated QoS values of a composite service (i.e., a solution) represented by the index *k*, $Q=(Q_1(SN),\ldots,Q_n(SN))$, where $Q_k(SN)$ is the estimated kth QoS characteristic of the composite service SN. The aggregation functions are presented in the table 2. Some aggregation functions are similar to those proposed by Zeng et al. [15] and Jaeger et al. [21]. In this table, the variable *a* represents the number of services involved for the computation. The variable $x_{ij}$ represents a selection variable. Note that the web services that only belong to the predicted path are considered for computation i.e. $i \in EP_{pred}$.

Table 2: QoS aggregation functions

| QoS characteristic | Aggregation function |
|---|---|
| Cost | $Q_k(SN) = \sum_{i \in EP_{pred}} \overset{k}{q}_{ij} * x_{ij}$ |
| Response time | $Q_k(SN) = \sum_{i \in EP_{pred}} \overset{k}{q}_{ij} * x_{ij}$ |
| Reliability | $Q_k(SN) = \prod_{i \in EP_{pred}} \overset{k}{q}_{ij} * x_{ij}$ |
| Availability | $Q_k(SN) = \prod_{i \in EP_{pred}} \overset{k}{q}_{ij} * x_{ij}$ |
| Encryption level | $Q_k(SN) = \min(\overset{k}{q}_{ij} * x_{ij}), \forall i \in EP_{pred}$ |
| Throughput | $Q_k(SN) = \min(\overset{k}{q}_{ij} * x_{ij}), \forall i \in EP_{pred}$ |
| Reputation | $Q_k(SN) = \frac{\sum_{i \in EP_{pred}} \overset{k}{q}_{ij} * x_{ij}}{a}$ |
| Composability | $Q_k(SN) = \frac{\sum_{i \in EP_{pred}} \overset{k}{q}_{ij} * x_{ij}}{a}$ |

As presented in table 2, availability and reliability are multiplicative QoS characteristics (aggregated as a product) and result in nonlinear functions. However, linear functions (aggregated as a summation) are easier to be processed. Therefore, similar to Zeng et al. [15], nonlinear functions are transformed into linear by applying a logarithm operation. For example, the aggregation function for reliability can be transformed to summation function as:

$$\log(Q_k(SN)) = \log(\prod_{i \in EP_{pred}} \overset{k}{q}_{ij}) = \sum_{i \in EP_{pred}} \log(\overset{k}{q}_{ij})$$

### 4.2.1 Utility function

If more than one QoS characteristics are subject to optimization, then an aggregated goal function is required to consider all the QoS characteristics. For this purpose, each candidate service $s_j \in S_i$ is associated with utility function $u_{ij}$. This function depends on the QoS characteristics' types i.e., positive or negative. For positive QoS characteristics, meaning that a higher value denotes a better QoS, like availability and reliability, the function needs to maximize the values, whereas, the values of the negative characteristics, like cost and response time, needs to minimize. In addition, QoS characteristics have different units of measurements. For example, reliability is a probability ratio and varies between 0 and 1 while response time is expressed in milliseconds by a positive number. To perform fair computation, and to allow uniform measurement of different QoS characteristics independent from their units and ranges, all the QoS characteristics are normalize by their average and slandered deviation. Furthermore, all the QoS characteristics are weighted by their importance.

***Definition 3: (Utility Function):*** Suppose that all eight QoS characteristics mentioned earlier are involved for optimization. The utility function is defined as follows [10]:

$$u_{ij} = \left( \sum_{k=1}^{2} \left( \frac{\overset{k}{\mu}_i - \overset{k}{q}_{ij}}{\overset{k}{\sigma}_i} \right) * W_k + \sum_{k=3}^{8} \left( \frac{\overset{k}{q}_{ij} - \overset{k}{\mu}_i}{\overset{k}{\sigma}_i} \right) * W_k \right) \quad (1)$$

Where $W_k$ is the weight assign for each QoS characteristics which defined by clients such that $W_k \in [0,1]$ and $\sum_{k=1}^{n} W_k = 1$. The index $k = 1 \ldots 8$ represents the different QoS characteristics. The first part of the equation i.e., the summation where $k = 1,2$ represents the normalization of the negative QoS characteristics, while the second represents the summation for the positive. $\mu_i$ and $\sigma_i$ are the average and standard deviation for the values of QoS characteristics of all candidates in service class $S_{i.}$

The selection variables $x_{ij}$ is used to determine whether a candidate service is selected for optimal composition or not. The value of $x_{ij}$ is either 0 or 1. 1 if the candidate service $s_{ij}$ is selected for the abstract service $as_{i,}$ while 0 if not. There is exactly one candidate service selected for each abstract service $as_i$ i.e., $\forall i, 1 \le i \le a, x_{i1} + x_{i2} + \ldots + x_{ib_i} = 1$ .

Based on the above, the selection problem can be modelled as follows:

$$\max( \sum_{i=1}^{1} \sum_{j=1}^{b_i} u_{ij} * x_{ij} )$$

Subject to the global QoS constraints

$Q_k(SN) \le \overset{k}{GS}$ , for negative QoS characteristics (price and response time)

$Q_k(SN) \ge \overset{k}{GS}$, for positive QoS characteristics (rest of QoS characteristics)

While keeping:

$$\sum_{j=1}^{b_i} x_{ij} = 1, \forall i \in \{1,\ldots,a\}, \text{and } x_{ij} = \begin{cases} 1, & if \quad s_{ij} \quad is \quad selected \\ 0, & if \quad s_{ij} \quad is \quad not \quad selected \end{cases}$$

### 4.3 Mapping to multi-dimensional multi-choice knapsack problem (MMKP)

***Definition 4: (MMKP):*** Suppose there are *N* object groups, each has $I_i (1 \le i \le N)$ objects. Each object has a profit ***p*<sub></sub>**$p_{ij}$ and requires resource $r_{ij} = (r1_{ij}, ..., rm_{ij})$. The amount of resources available in the knapsack is $R = (R1,..., Rm)$. MMKP is to select exactly one object from each object group to be placed in the knapsack so that the total profit is maximized while the total resources used are less than the resources available.

On the other hand, QoS selection aims to select exactly one candidate from each service class such that the entire QoS value of the composition is optimized while QoS requirements defined by clients are satisfied.

Building on the similarity between these two problems, the selection problem can be mapped to MMKP as follows:

- The knapsack is represented by the composition.
- Each service class represents an object group.
- Each candidate in a service class represents one object in a group.
- Each utility function $u_{ij}$ represents a profit $p_{ij}$ and can be calculated using equation (1).
- The QoS characteristics $q_{ij}$ of a candidate $s_j$ represent the required recourse $r_{ij}$ of an object.
- The QoS global constraints *GS* are considered as the resources available in the knapsack *R*.
- The algorithm is to select exactly one candidate from each service class such that the entire QoS value of the composition is optimized while QoS requirements defined by clients are satisfied. However, the MMKP requires that the total resources are less than the recourse available. But in the selection problem, the total QoS characteristics are required to be either less (for negative characteristics) or greater (for positive) than the global QoS constraints. Therefore, positive characteristics are needed to be transformed into negative. To do so, the values of positive characteristics are multiplied by -1.

The service selection problem can be formulated mathematically as follows, where *O* represents the objective function:

maximize $O = (\sum_{i=1}^{1} \sum_{j=1}^{b_i} u_{ij} * x_{ij})$

Subject to the QoS global constraints

$Q_k(SN) \le \overset{k}{GS} \quad , \forall i = 1,..., a, \forall j = 1,..., b_i, \forall k = 1,...n$

While keeping

$\sum_{j=1}^{b_i} x_{ij} = 1, \forall i \in \{1,..., a\}$, and

$$x_{ij} = \begin{cases} 1, & if \quad s_{ij} \quad is \quad selected \quad for \quad class \quad S_i \\ 0, & if \quad s_{ij} \quad is \quad not \quad selected \end{cases}$$

MMKP is known as NP-hard [23]. Due to its high computational complexity, approaches that deliver exact optimal solutions are inappropriate for real time decision-making applications. Especially in our scenario where a quick response for a workflow instance is very important. Thus, heuristic represents a novel approach. Therefore, new adapted heuristic algorithms are proposed to reduce the computation efforts.

### 4.4 Heuristic algorithms for the selection problem

Building on the analogy between the MMKP and the selection problem, our approach is to adapt heuristics that are known to be efficient for solving MMKP, and apply it to solve the selection problem.

For solving the MMKP, a constructive and complementary search approach is developed by Hifi et al. [24]. In this approach, the constructive procedure CP is applied to generate a feasible solution, while the complementary procedure CCP is used to improve the quality of the solution generated from CP. According to the authors, the experiments result shows that the algorithms generate high-quality solutions within small computing times. The algorithms are adapted here to solve the selection problem. These algorithms are chosen because of their ability to generate quality solutions, reduce the computation efforts resulted from the problem, and finally can be easily applied to solve the selection problem.

### 4.5 Solving the selection problem using CP and CCP

First, we introduce the representation of a solution along with some notions:

A solution (*SN)* for the selection problem can be represented as illustrated in Fig 3.

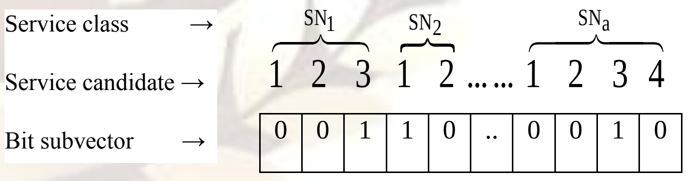

**Figure 3: binary representation for a solution**

For each service class $S_i$, one and only one candidate service $s_j$ is selected, i.e., $x_{ij} = 1$ if the $j^{th}$ candidate service *s* of the $i^{th}$ service class $S_i$ has been selected, otherwise $x_{ij} = 0$. A feasible solution is such that:

$$\forall k \in \{1,..,n\}, \sum_{i=1}^{a} \sum_{j=1}^{bi} \overset{k}{q}_{ij} * x_{ij} \le \overset{k}{GC}, \forall i \in EP_{pred}$$

Note that the feasibility of the predicted execution path is only checked, i.e., $iff : \forall i \in EP_{pred}$. The predicted execution path $EP_{pred}$ is determined using the data mining method introduced later.

For the solution *SN*, there are two distinguished states: feasible state (FS); if the solution *SN* does not violate the amount of available global QoS constraints, and unfeasible state (US); if the solution *SN* violates at least one or more global QoS constraints.

### 4.5.1 Initial feasible solution

The initial feasible solution is obtained using the CP. The CP is a greedy approach with DROP and ADD phases to generate a feasible solution. The steps involved in the algorithm are introduced as follows:

- Calculate the pseudo-utility ratio (*ut*) for each candidate service s using this equation:

  $$ut_{ij} = u_{ij} / \sum_{k=1}^{n} \overset{k}{GS} * \overset{k}{q}_{ij}, \forall k \in \{1,..., n\}$$

  Where $u_{ij}$ is the utility function defined in (1), and $\overset{k}{GS}, \overset{k}{q}_{ij}$ are the global QoS constraints and the QoS characteristics values respectively.

- Select the candidate *s* from each service class $S_i$, $i \in \{1,.., a\}$ which has the maximum pseudo-utility ration $ut_{ij}$.

- Check the state of the obtained solution *SN*: if feasible state (FS), then CP terminates; else (DROP phase), it determines the most violate constraint $\overset{k_o}{GS}$. With respect to $\overset{k_o}{GS}$, it selects the service class $S_{i_o}$ corresponding to the fixed candidate service $s_{i_o}$ having the largest QoS value $\overset{k_o}{q}_{i_o j i_o}$ all over the fixed candidate services and regarding the most violated constraint.

- (ADD phase) Swap the selected candidate service with another candidate s from the same service class $S_{i_o}$ .

- Check the feasibility of new obtained SN, if US; select the lightest candidate $s'_{i_o}$ of the current service class $S_{i_o}$ which in turn is considered as the new selected candidate service.

- Iterate until an FS or the smallest unfeasibility amount is obtained.

### 4.5.2 Improving the initial feasible solution

To improve the QoS values of the initial feasible solution *SN* obtained by CP, CCP is applied. It tries to iteratively improve the solution *SN* by applying:

A. A local swap strategy for selected candidate services that belongs to *SN*, called old candidates.

B. A replacement stage that replaced the old candidate to another new one, called new candidate, selected from the same service class. Each replacement between an old and new candidate is authorize if and only if the obtained solution *SN* realizes a *FS*, i.e., maintains the feasibility of *SN*.

The procedure steps are introduced as follows:

- Apply CP to obtain initial feasible solution.
- Initially, set the best solution equal to the solution obtained by CP.
- Start the conditional loop by performing a local swap search strategy procedure in order to improve the initial solution.
- If the obtained solution (obtained after performing the local swap strategy) realizes a better solution value compared to the initial one; then set the best current solution equal to the obtained one.
- Repeat the loop until the condition is true.

The steps of the Local swap search procedure are introduced as follows:

Step 1: Initialize the best candidate service to swap:

1.1 $value \leftarrow u_{iSN_i}$, where $u_{iSN_i}$ is the utility of the old selected candidate $s_i$ in the ith service class $S_i$ to be swapped.

1.2 $k_i \leftarrow SN_i$, where $k_i$ is a selected candidate service in $S_i$ service class to be swapped.

Step 2: Perform the exchange if authorized:

2.1 perform the exchange if the there is a new candidate service *s* that has larger QoS value than the old candidate and, at the same time, realizes a FS.

2.2 return the best candidate service $k_i$ to be swapped.

## 4.6 Execution path prediction

Organizations adopt Workflow Management Systems (WFMS) to define, manage and execute their business processes

WFMS store the data generated from the execution of workflows in logs. The data stored in databases (logs) are rich with concealed information that can be used for making intelligent business decisions [25]. On possible way to reveal this valuable information is by applying data mining algorithms on these logs.

Apart from the QoS-aware service composition, Cardoso [26] emphasizes the importance of QoS management for workflows and organizations. He focusses on predicting the QoS of workflows before they executed or during the execution. To this aim, the author proposes a novel method, based on data mining techniques, that allows predicting with high level of accuracy the QoS of workflows. Their method is adapted here for the purpose of predicting, at runtime, the sequence of web services that will be executed during a composition's execution.

### 4.6.1 Composite service scenario

This section describes the scenario that will be used throughout this paper to explain the method of predicting the execution path. The scenario is the same used by Cardoso [26].

A major bank has decided to adopt a WFMS in order to support its business processes. The business processes (composite services) supported by the bank are developed using web service composition technology. Each function of the composite services can be accomplished by a single outsourced web service.

One of the composite services supplied by this bank is the loan composite service illustrated in Fig 4. It is composed of 13 web services which are connected using sequential (services are executed in sequence order) and conditional structures (among all the branches, only one branch is executed).

In this composite service, a client (i.e., a bank loan's requester) is required to fill electronic from for requesting a loan. The data provided by the client are forwarded to Check Loan Type web service to determine the loan types. Based on its type, the request is then forward to one of the three services: Check Home Loan, Check Educational Home Loan, or Check Car Loan. The request can be: accepted or rejected or approved conditionally in the case of a home loan. Approve (Reject) Home Loan, Approve (Reject) Educational Loan, and Approve (Reject) Car Loan are the web services in charge of accepting (rejecting) a loan request. The result of the loan request is then e-mailed to the client. Finally, the loan application data is stored in a database by the Archive Application web service.

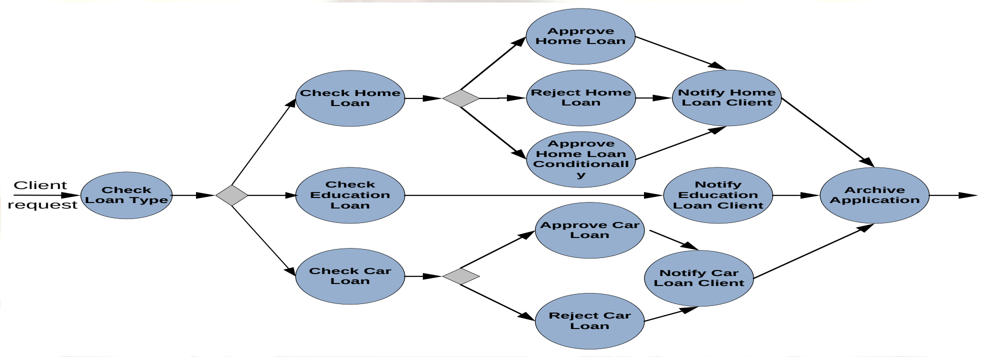

**Figure 4: Bank loan composite service [26]**

### 4.6.2 Composition logs

In PAIS (Process Aware Information System) such as WFMS, the data generated from the execution of business processes are recorded into so-called execution logs. During the execution of a composite service, WFMS stores data including real time information describing the execution and the behaviour of the composite service, web services, and instances. Table 3 illustrates an example of a composition log. The data stored in these logs are rich with concealed information. One useful and important piece of knowledge that can be extracted from these logs is the subset of the web services that will potentially be executed by composition instance.

### 4.6.3 Path prediction method

The adapted data mining method is composed of three steps, the first two are similar to Cardoso, while the third is added to the purpose of runtime predication, and carried out at runtime.

#### 4.6.3.1 Extending the composition log

Additional data are required to be recorded in logs to carry out execution path prediction. These data are runtime generated information indicating the input (output) values parameters passed (received) to (from) web services and their types. These values are generated at runtime during the execution of composition instances. Each 'parameter/value' entry as a data type, a name, and a value, (for example, int loannumber=12323). To store such information, the current composition logs need to be extended. Furthermore, an extra field needs to be added to the log in order to store execution path information; which describing the path that has been taken during the execution of a composite service (i.e., the web services that have been executed). Table 4 illustrates an example of extended composition log.

**Table 3: Composition log**

| Date | Composition | Instance | Web service | Service instance | |
|---|---|---|---|---|---|
| 11:57 23-02-2011 | Loan Application | LA112 | RejectHomeLoan | RHL01 | … |
| 11:58 23-02-2011 | Loan Application | LA112 | NotifyUser | NU22 | … |
| 12:05 23-02-2011 | Insurance Claim | IC186 | CheckClaim | FCR54 | … |
| … | … | … | … | … | … |

**Table 4: Extended composition log**

| .. | Instance | Web service | Service instance | .. | Parameter/value | EP |
|---|---|---|---|---|---|---|
| .. | LA112 | RejectHomeLoan | RHL01 | .. | int loannumber=12323; string email='ahmad@yahoo.com' string loantype='home-loan' | FillLoanRequest,CheckLoanType,CheckHomeLoan,RejectHomeLoan |
| .. | LA112 | Archive Application | NU22 | .. | string tel='1726354'; string email='ali@hotmail.com' | FillLoanRequest,CheckLoanType,CheckHomeLoan,RejectHomeLoan,NotifyHomeLoanClientArcheiveApplication |
|  | … | … | … | .. | … | … |

### 4.6.3.2 Learning phase (Offline)

Once enough data are recorded in a composition log, data mining instances need to be constructed by extracting the input/output values of web services as well as the path information from the log. The extracted data are converted to a suitable format to be processed by machine learning algorithms. Each instance is characterized by the values of six input/output parameters of web services and associated with a class, namely EP, indicating the path that has been followed during the execution when the parameters of web services have been assigned to a specific value set. In the bank loan composite service, there are six possible alternative paths a class EP can take: $EP_{1,}$ $EP_2$ … $EP_6$. In summary, the instance structure is as follows:

income, loan type, loan amount, loan years, name, SSN , [EP]

The parameter income, loan amount, loan years and SSN are numeric, whereas the attributes loan type and name are nominal. For example, loan-type can take the finite set of values: home_loan, education_loan, and car_loan.

Once enough instances are created, it will constitute inputs to a machine learning algorithm to establish a relationship between the input/output parameters and the paths taken at runtime. A set of classified instances is taken by a learning schema to learn a way of classifying unseen instances. Since the class EP of each instances is provided, we uses supervised learning [25].

Note that, in our approach, the algorithm will be trained offline so the performance of the approach does not be affected by the computation time needed for training the algorithm.

Different machine learning methods can be employed to carry out path prediction. Cardoso (2008)[25] conducts a set of experiments using Naïve Base (NB), J48, and Sequential Minimal Optimization (SMO) methods with and without the Multiboost (MB) method.

J48 algorithm is Weka's (2004) implementation of the C4.5 [27] decision three learner. Naïve Bayes (NB) classifier technique is based on the so-called Bayesian theorem. SMO [28] is a fast method to train SVM (Support Vector Machines) [29]. MB [30] is an improved meta learning algorithm of AdaBoost [31].

Based on Cardoso's experiments, the results indicate that the Multiboost Naïve Base (MB NB) approach is the data mining algorithm that yields the best path prediction accuracy results [25]. Therefore, this work employs the MB NB as well.

### 4.6.3.3 Path prediction phase (at runtime)

Once the data is formatted and analyzed by a learning method, it is now ready for classifying unknown classes, i.e., predict the path that will be followed during the execution.

At runtime, a client (i.e., a service requester) for a bank loan is required to provide data including personal data and data describing the condition of the service being requested. For example, income, loan type, loan amount, loan years, Name, and SSN are examples of such data. Then, the data needed for prediction are collected and formatted, and then input to a classifier to classify into target classes, i.e., composition paths.

The output of this step is the predicted probability of a given execution path $EP_i$ will potentially be followed during the execution of the bank loan composite service. This important information is then utilized by the optimization algorithms in order to only optimize the predicted execution path.

## 5. CONCLUSION

We have proposed an innovative approach that computes the optimization by considering only the path that potentially will be followed during the execution of a composition. Organizations can use our approach to build their business processes. By using our approach: (i) it is expected to always generate business process that deliver the best possible QoS, at the same time, meet global QoS constraints, (ii) it is expected to generate business processes in reasonable time. We are planning to evaluate the approach using simulation software.

## REFERENCES

[1] G. Baryannis, M. Carro, O. Danylevych, S. Dustdar, D. Karastoyanova, K. Kritikos, L. Philipp, F. Rosenberg, and B. Wetzstein, Overview of the State of the Art in Composition and Coordination of Services, S-CUBE consortium, Tech. Rep., July 2008. [Online]. Available: http://www.s-cube-network.eu/

[2] M.O. Hilari, *Quality of Service (QoS) in SOA Systems. A Systematic Review*, Master thesis, Universitat Politècnica de Catalunya, 2009.

[3] F. Rosenberg, P. Leitner, A. Michlmayr, P. Celikovic, and S. Dustdar, Towards Composition as a Service - A Quality of Service Driven Approach. *Proc. 2009 IEEE International Conference on Data Engineering*, IEEE Computer Society, 2009, 1733-1740.

[4] S. Dustdar, and M.P. Papazoglou, Services and Service Composition - An Introduction, *it - Information Technology, 50*, 2008, 086 - 092.

[5] D. Ardagna, and B. Pernici, Global and Local QoS Guarantee in Web Service Selection, in C. Bussler and A. Haller (Eds.), *Business Process Management Workshops*, 3812 (Heidelberg: Springer Berlin 2006) 32-46.

[6] R. Ukor, and A. Carpenter, On Modelled Flexibility and Service Selection Optimisation, *Proc. 9th Workshop on Business Process Modeling, Development and Support*, Montpellier, 2008, 335.

[7] H. Schonenberg, R. Mans, N. Russell, N. Mulyar, and W. Aalst, Process Flexibility: A Survey of Contemporary Approaches, in W. Aalst, J. Mylopoulos, N. M. Sadeh, M. J. Shaw and C. Szyperski (Eds.), *Advances in Enterprise Engineering I*, 10 (Heidelberg: Springer Berlin 2008) 16-30.

[8] M.R. Garey, and D.S. Johnson, *Computers and Intractability: a Guide to the Theory of NP-Completeness* ( New York: W. H. Freeman & Co., 1979).

[9] M. Jaeger, G. Muhl, and S. Golze, QoS-Aware Composition of Web Services: An Evaluation of Selection Algorithms, in R. Meersman and Z. Tari (Eds.), *On the Move to Meaningful Internet Systems 2005*, 3760 (Heidelberg: Springer Berlin 2007) 646-661.

[10] T. Yu, Y. Zhang, and K.-J. Lin, Efficient algorithms for Web services selection with end-to-end QoS constraints, *ACM Transactions on the Web (TWEB), 1(1)*, 2007, 6.

[11] G. Canfora, M.D. Penta, R. Esposito, and M.L Villani, An approach for QoS-aware service composition based on genetic algorithms, *Proc. 2005 conference on Genetic and evolutionary computation*, Washington, ACM, 2005, 1069-1075.

[12] R. Wang, C.-H. Chi, and J. Deng, A Fast Heuristic Algorithm for the Composite Web Service Selection, in Q. Li, L. Feng, J. Pei, S. Wang, X. Zhou and Q.-M. Zhu (Eds.), *Advances in Data and Web Management*, 5446 (Heidelberg: Springer Berlin 2009) 506-518.

[13] F. Lécué, Optimizing QoS-Aware Semantic Web Service Composition, in A. Bernstein, D. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, and K. Thirunarayan (Eds.), *The Semantic Web - ISWC 2009*, 5823 (Heidelberg: Springer Berlin 2009) 375-391.

[14] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q.Z Sheng, Quality driven web services composition, *Proc.12th international conference on World Wide Web*, Budapest, ACM, 2003, 411-421.

[15] L. Zeng, B. Benatallah, A.H.H Ngu, M. Duma, J. Kalagnanam, and H. Chang, QoS-aware middleware for Web services composition, *IEEE Transactions on Software Engineering, 30(5)*, 2004, 311-327.

[16] R. Ukor, and A. Carpenter, Flexible Service Selection Optimization Using Meta-Metrics, *Proc. 2009 Congress on Services – I*, IEEE Computer Society , 2009, 593-598.

[17] S. Neelavathi, and K. Vivekanandan, An Innovative Quality of Service (QOS) based Service Selection for Service Orchrestration in SOA, *International Journal of Scientific and Engineering Research, 2(4)*, 2011.

[18] O.K. Qtaish, and Z.B. Jamaludin, QoS criteria for distinguishing the competing web services, *Proc. 2011 International Conference on Data Engineering and Internet Technology (DEIT)*, Bali, IEEE Computer Society, 2011.

[19] K. Lee, J. Jeon, W. Lee, S.-H Jeong, and S.-W Park, QoS for Web Services: Requirements and Possible Approaches, W3C Web Services Architecture Working Group, Tech. Rep., November 2003. [Online]. Available: http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/ last accessed Feb, 2012.

[20] Mani, and A. Nagarajan, Understanding quality of service for Web services, IBM developerWorks, Tech. Rep., January 2002. [Online]. Available: http://www.ibm.com/developerworks/webservices/library/ws-quality/index.html last accessed Feb, 2012.

[21] M.C. Jaeger, G. Rojec-Goldmann, snd G. Muhl, QoS aggregation for Web service composition using workflow patterns, *Proc. Enterprise Distributed Object Computing Conference, Eighth IEEE International*, Washington, IEEE Computer Society, 2004, 149-159.

[22] Z. Guoping, Z. Huijuan, and W. Zhibin, A QoS-Based Web Services Selection Method for Dynamic Web Service Composition, *Proc. 2009 First International Workshop on Education Technology and Computer Science*, Hubei, IEEE Computer Society, 2009, 832-835.

[23] S. Martello, and P. Toth, Algorithms for knapsack problems, in S. Martello, G. Laporte, M. Minoux, and C. Ribeiro (eds.), *Surveys in Combinatorial Optimization*, 31 (Amsterdam: Annals of Discrete Mathematics 1987) 213–258.

[24] M. Hifi, M. Michrafy, and A. Sbihi, Heuristic algorithms for the multiple-choice multidimensional knapsack problem, *Journal of the Operational Research Society*, *55*, 2004, 1323–1332.

[25] S. Sumathi, and S. Esakkirajan, *fundamentals of relational database management systems* (Heidelberg: Springer Berlin 2007).

[26] J. Cardoso, Applying Data Mining Algorithms to Calculate the Quality of Service of Workflow Processes, in P. Chountas, I. Petrounias and J. Kacprzyk (Eds.), *Intelligent Techniques and Tools for Novel System Architectures*, 109 (Heidelberg: Springer Berlin 2008) 3-18.

[27] J.R. Quinlan, *C4.5: programs for machine learning* (San Francisco: Morgan Kaufmann Publishers Inc., 1993).

[28] J.C. Platt, Fast training of support vector machines using sequential minimal optimization, in B. Scholkopf, C. J. C. Burges and A. J. Smola (Eds.), *Advances in kernel methods*, *(*MIT Press 1999) 185-208.

[29] C. Cortes, and V. Vapnik, Support-vector networks, *Machine Learning, 20(3*), 1995, 273-297.

[30] L. Todorovski, and S. Džeroski, Combining Multiple Models with Meta Decision Trees, in D. Zighed, J. Komorowski and J. Zytkow (Eds.), *Principles of Data Mining and Knowledge Discovery*, 1910 (Heidelberg: Springer Berlin 2000) 69-84.

[31] Y. Freund, and R.E. Schapire, A Short Introduction to Boosting, *Journal of Japanese Society for Artificial Intelligence, 14(5)*, 1999, 771-780.