Set-based Bi-level Optimisation for QoS-aware Service Composition in Ubiquitous Environments

Nebil Ben Mabrouk, Nikolaos Georgantas, Valerie Issarny

To cite this version:

Nebil Ben Mabrouk, Nikolaos Georgantas, Valerie Issarny. Set-based Bi-level Optimisation for QoS-aware Service Composition in Ubiquitous Environments. Proceedings of the 22nd IEEE International Conference on Web Services (ICWS), Jun 2015, New York, United States. 10.1109/ICWS.2015.14 . hal-01274038

HAL Id: hal-01274038
https://hal.inria.fr/hal-01274038

Submitted on 15 Feb 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Set-based Bi-level Optimisation for QoS-aware Service Composition in Ubiquitous Environments

Nebil Ben Mabrouk, Nikolaos Georgantas, and Valérie Issarny
Inria Paris-Rocquencourt, Domaine de Voluceau, Le Chesnay, 78150, France.
E-mail: nebil.ben-mabrouk,nikolaos.georgantas,valerie.issarny@inria.fr

***Abstract*—Service composition is a widely used method in ubiquitous computing that enables accomplishing complex tasks required by users based on elementary (hardware and software) services available in ubiquitous environments. To ensure that users experience the best Quality of Service (QoS) with respect to their quality needs, service composition has to be *QoS-aware*. Establishing QoS-aware service compositions entails efficient service selection taking into account the QoS requirements of users. A challenging issue towards this purpose is to consider service selection under global QoS requirements (i.e., requirements imposed by the user on the whole task), which is of high computational cost. This challenge is even more relevant when we consider the dynamics, limited computational resources and timeliness constraints of ubiquitous environments.**

To cope with the above challenge, in this paper we present QASSA, an efficient service selection algorithm that provides the appropriate ground for QoS-aware service composition in ubiquitous environments. The contribution of QASSA is three-fold. First, it formulates service selection under global QoS requirements as a set-based optimisation problem, benefiting from recent proposals in the domain of multi-objective optimisation. Second, QASSA resolves this problem in an efficient way using clustering techniques, namely the K-Means algorithm. Third, QASSA is devised in two versions, viz., centralised and distributed versions, so that it can be executed on top of centralised and decentralised infrastructures in ubiquitous environments. Results of experimental studies are presented to illustrate the timeliness and optimality of QASSA.

I. Introduction

Ubiquitous (computing) environments enable integrating and composing, on the fly, services that are offered by (hardware and software) resources available in the environment in order to fulfil complex tasks required by users. Nevertheless, fulfilling the user's tasks from the functional point of view only is not enough to gain user satisfaction. Users further require a certain Quality of Service (QoS) when exerting their tasks. For this reason, a lot of research efforts in ubiquitous computing have been devoted to the composition of services under the user's QoS requirements, which is known as QoS-aware service composition. QoS-aware service composition is a broad topic. At the core of this topic is the issue of QoS-aware service selection, which allows determining services available in ubiquitous environments and able to meet the user's QoS requirements. The problem arises when dealing with complex user tasks formed of multiple (abstract) activities, and each activity can be achieved using several services that are functionally equivalent, but providing different QoS levels. The question to be asked is then: "*what are the services that should be selected for each activity in the user's task in order to meet the user's QoS requirements and produce the highest QoS?*"

Addressing the above question is even more complicated when considering the challenges entailed by the characteristics of ubiquitous computing. These challenges are mainly about: (i) timeliness (i.e., achieving service composition in a timely manner with respect to on-the-fly interaction with users in ubiquitous environments), (ii) considering QoS variations at run-time, notably during the time elapsing between services' selection and services enactment, (iii) adaptation support, and (iv) supporting both centralised and distributed infrastructures in ubiquitous environments.

To cope with the above challenges, we introduce an efficient service selection algorithm called QASSA (Qos-Aware Service Selection Algorithm). Compared to existing solutions, QASSA presents the following contributions:

1) It models QoS-aware service selection under global QoS requirements in a novel way benefiting from recent mathematical proposals that define multi-objective optimisation as a set-based selection problem [1];
2) Thanks to its underlying model, QASSA resolves QoS-aware service selection efficiently using clustering techniques, thus executing in a timely manner (on top of both resource-enabled and resource-limited devices) while achieving a nice optimality (further details are given in Section IV);
3) Based on clustering techniques, QASSA can distinguish several classes of services that represent different tradeoffs between QoS properties, hence enabling fine-grained management of these tradeoffs with respect to user preferences;
4) QASSA selects several alternative service compositions (instead of only one). That is, it selects several interchangeable services for each activity in the user task, thus enabling service substitution (and accordingly adaptation support) at run-time;
5) QASSA is devised in both centralised and distributed fashions, which makes it suitable for both infrastructure-enabled and infrastructure-less ubiquitous environments.

Next, we start by presenting the design rationale of QASSA, its underlying problem definition, as well as its both phases, viz., local and global selection phases (Section II). After that, we introduce a distributed version of QASSA that is able to execute in infrastructure-less ubiquitous environments (Section III). Finally, we give the results of experimental studies illustrating the efficiency of QASSA in terms of timeliness and optimality (Section IV), and we conclude in Section VI.

II. The QASSA Algorithm

A. Notations and Basic Assumptions

QASSA is initialised by taking as input a user request R, which is defined as a quadruple $R = (T, U, P, W)$, where T refers to the required task and U refers to global QoS constraints $U = \langle u_1, .., u_n \rangle$ imposed by the user on a set of QoS properties $P = \langle p_1, .., p_n \rangle$. For each constraint, the user has to specify the relative importance of its associated QoS property by giving a set of weights $W = \langle w_1, .., w_n \rangle$, where w_i is the weight of QoS property p_i. It is worth noting that the sum of all the weights must be equal to 1, i.e., $\sum_{i=1}^{n} w_i = 1$.

For a user task T, its structure is specified as a set of activities $T = \langle A_1, ..A_z \rangle$ coordinated by composition patterns. To each activity A_i in T is associated a set of concrete service candidates $S = \{s_{i,1}, s_{i,2}, .., s_{i,m_i}\}$ that are able to realise A_i. Each service $s_{i,k}$ ($1 \leq k \leq m_i$) is represented by its QoS vector $QoS_{s_{i,k}} = \langle q_1, .., q_n \rangle$, where q_j is the advertised value of QoS property p_j ($1 \leq j \leq n$).

In this paper, we do not deal with the way QoS is specified, rather we recall the QoS model presented in our previous work [2] to define QoS properties and preferences, as well as QoS aggregation formulae. Additionally, we consider stateless services that can be bound to one or more abstract activities in the composition. The QoS values advertised by these service are supposed to be specified by the service providers based on the previous executions of the services.

B. Design Rationale

QoS-aware service selection algorithms fall under two broad classes with respect to their selection techniques. On the one hand, *local selection* (i.e., greedy selection) proceeds by selecting the best service in terms of QoS for each activity in the user task separately. This technique has a low computational cost but it can not guarantee meeting global QoS requirements. On the other hand, *global selection* covers the scope of the whole composition and ensures meeting global QoS requirements. However, it is of high computational complexity.

QASSA combines local and global selection techniques in order to handle the complexity of service selection under global QoS requirements. Specifically, QASSA follows the bi-level optimisation model [3], which is a hierarchy of two optimisation problems (upper-level or leader, and lower-level or follower problems). Each problem is optimised separately without considering the objective of the other one. However the decision made at the upper-level problem affects the objective space of the lower-level as well as the decision space.

In accordance with the bi-level optimisation model, QASSA proceeds through two main steps: (1) *local selection* (representing the upper-level optimisation problem), which aims at selecting services with the highest QoS for each activity in the user task, and (2) *global selection* (representing the lower-level optimisation problem), which aims at selecting near-optimal compositions of services resulting from the local selection. QASSA further selects several alternative near-optimal compositions. Indeed, selecting only one service composition brings about several shortcomings such as the lack of choices for the user, the overload of *hot services* (i.e., services with high QoS) [4], and the lack of adaptation support [5] (i.e., deferred final selection and dynamic binding at run-time).

C. Local Selection Phase

We express the local selection as a multi-objective optimisation problem, since it involves numerous optimisation objectives, such as the cost and the delay of services. Additionally, we focus on resolving this problem while producing several solutions (and not a single solution) to enable the global selection phase and allow adaptation at run-time.

1) Problem definition: A recent proposal by Zitzler et al. [1] defines multi-objective optimisation as a set problem having as goal to identify the *Pareto optimal set* (i.e., the best solution set) among several solution sets, each set reflecting a specific tradeoff between the optimisation objectives [6]. Determining the Pareto optimal set is typically exponential in the size of the problem instance [7]. Therefore, resolving set-based multi-objective optimisation is often reduced to identifying a good *Pareto set approximation*.

In accordance with Zitzler's proposal, we define the local selection problem as a set-based multi-objective optimisation. Consider the optimisation of the QoS vector $P = \langle p_1, ..., p_n \rangle : S \rightarrow \mathbb{R}^n$ where all QoS properties p_i are, without loss of generality, to be maximised. Here, S denotes the feasible set of solutions, i.e., the set of service candidates of a given activity in the user task. A single service $s \in S$ will be denoted as a decision vector or solution ($s = \langle q_1, ..., q_n \rangle$) where $q_i = p_i(s)$.

We define the search space $\mathbb{S}$ by the set of feasible sets of services (and not single services). In the context of QoS optimisation, an element (i.e., a service-set) in $\mathbb{S}$ is called *QoS Class*, and denoted $QC \in \mathbb{S}$. A QoS class represents a set of services having *roughly* the same QoS and reflecting the same tradeoff between QoS properties.

To enable the comparison of QoS classes, a *set preference relation* must be defined over $\mathbb{S}$. A set preference relation provides the information on the basis of which the selection is carried out; it says whether a QoS class is better (in terms of QoS) than another one, or not. The set preference relation can be defined in terms of a *quality indicator* (such as the hypervolume indicator [7]). The quality indicator is a function that assigns, to each solution-set, a scalar value reflecting its quality according to a particular goal, i.e., a fitness function defined over sets. In this paper, we need to define a quality indicator I_q that is specific to our QoS optimisation problem (see Equation 2). Our objective is then to find a solution-set that maximises the value of I_q as defined below (the operator *argmax* returns the QoS class for which I_q attains its maximum value):

$$argmax\Big(I_q(\mathrm{QC})\Big) \qquad where\ QC \in \mathbb{S} \tag{1}$$

Solving this problem consists in determining $\mathbb{S}$ and its underlying QoS Classes $QC \in \mathbb{S}$, as well as defining the quality indicator I_q. Towards this purpose, we propose investigating clustering techniques, notably the K-means algorithm. Clustering techniques allow for grouping a set of data into several clusters with respect to given criteria. If we apply the same principle to our purpose (i.e., QoS-aware service selection), we can group service candidates associated with an activity into several clusters according to their QoS values. Each cluster includes services having roughly the same QoS. We can further define a quality indicator on these clusters of services based on their respective QoS properties. Next, we show how to solve the local selection problem (defined as a set-based multi-objective optimisation) using K-means, further introducing the formal definition of QoS Class and quality indicator.

2) Local selection in QASSA: To select a set of services providing high values for all QoS properties, we perform one-dimensional clustering applied n times (once per QoS property). That is, we cluster service candidates (i.e., associated with each activity in the composition) for each QoS property separately. Thus, for each QoS property p_j we obtain several clusters $C = \langle c_{1,j}, .., c_{g,j} \rangle$, going from the cluster $c_{1,j}$ of services having the lowest values of p_j to the cluster $c_{g,j}$ of services with the highest values for the same property (respecting the definitions of positive and negative QoS properties). After that, by considering the intersection of the clusters with the highest values associated with each QoS property, we obtain the service-set providing high values for all QoS properties jointly. To formally explain

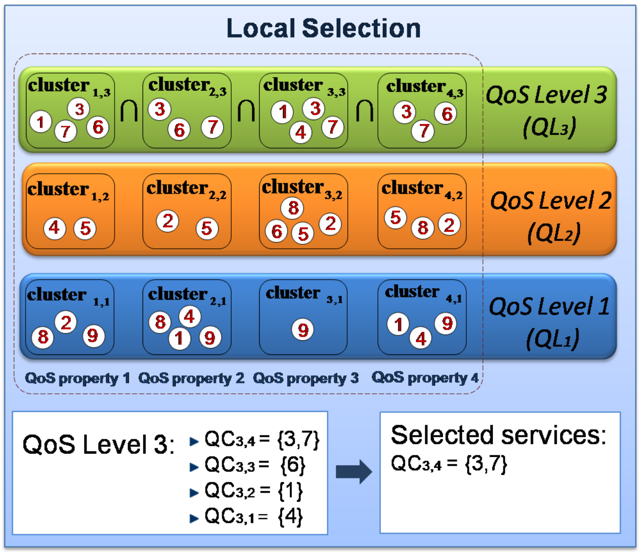


Fig. 1. Local selection in QASSA

the local selection in QASSA, we introduce the concept of *QoS Level* and *QoS Class*.

The concept of QoS Level is used to group together service clusters having roughly the same quality level for all QoS properties. The number of QoS levels corresponds to the number of clusters for each QoS property denoted g. For instance, in Figure 1 we cluster candidate services into 3 clusters, thus obtaining 3 QoS levels (1, 2 and 3) corresponding to the clusters of services having respectively 'low', 'medium' and 'high' QoS values for all QoS properties. As we have 4 QoS properties, each QoS level comprehends 4 clusters. Below, we formally define the QoS level concept.

Definition 1: Given a set of QoS properties $P = \langle p_1, .., p_n \rangle$ and a set of services $S = \{s_{i,1}, s_{i,2}, .., s_{i,m_i}\}$ associated to activity A_i and grouped into g clusters $\langle c_{1,j}, .., c_{g,j} \rangle$ for each QoS property p_j (where $c_{1,j}$ is the cluster of services with the lowest values of p_j and $c_{g,j}$ is the cluster of services with the highest values of p_j), we define a QoS level $QL_l = \{c_{l,1}, .., c_{l,n}\}$ as the set of clusters associated with each QoS property p_j and having the same level l $(1 \leq l \leq g)$.

As stated above, a QoS level QL_l is used to group clusters with the same level l together, thus we can perform their intersection and determine services with QoS values in this level. In particular, we are interested in the best QoS level $QL_g = \{c_{g,1}, .., c_{g,n}\}$ which groups clusters with the highest QoS values. The intersection of these clusters yields services with the highest QoS values for all QoS properties. However, if the intersection produces an empty set, we investigate other combinations (i.e., intersections) of clusters within QL_g. To do so, we introduce the concept of QoS Class, which represents different intersections of clusters within a given QoS level. The QoS class concept is formally defined as follows.

Definition 2: Given a QoS level $QL_l = \{c_{l,1}, .., c_{l,n}\}$, we define a QoS class $QC_{l,e}$ $(1 \leq e \leq n)$ as the intersection of e clusters among QL_l. Consequently, a QoS level QL_l comprehends several QoS classes (e.g., $\{QC_{l,1}, .., QC_{l,n}\}$).

Literally, a QoS class $QC_{l,e}$ represents the set of services having exactly e QoS properties out of n at the QoS level l. According to this, the QoS class $QC_{g,n}$ groups the best set of services in terms of QoS, since they have all their n QoS properties in the highest QoS level QL_g. If $QC_{g,n}$ is an empty set (i.e., there are no services with high values for all QoS properties), we try to find the next best QoS class in terms of QoS (e.g., $QC_{g,n-1}$).

Nevertheless, we may obtain several QoS classes having the same level and the same number of QoS properties (e.g., selecting $n-1$ QoS properties out of n). To determine the best QoS class, we use the quality indicator I_q (already introduced in Section II-C1). I_q is defined based on the level l and the number of QoS properties e of the considered QoS class, as well as the weights w_j associated with these QoS properties. That is, the quality indicator I_q of the QoS class $QC_{l,e}$ is higher (i.e., it includes services with better QoS) when: (i) it is associated with a QoS level QL_l of a higher level l, (ii) it comprehends a higher number of QoS properties e in that level, and (iii) the weights w_j associated with these QoS properties are more important for the user. I_q is formally defined as follows:

$$I_q(QC_{l,e}) = l \times e \times \sum_{j=1}^{e} w_j \qquad w_j \in \{W : c_{l,j} \in QC_{l,e}\} \tag{2}$$

Finally, it is worth noting that QASSA implements the local selection using a variant of K-means called K-means++ [8], which takes as input only the number of clusters (in opposition to K-means which takes as input the number of centroids and their initial coordinates). The number of clusters is determined beforehand by learning from the previous executions of the considered activity using existing techniques in the literature (viz., the Davies-Bouldin index [9]) that determine whether a given integer g is the most appropriate number of clusters for classifying a fixed data set. The Davies-Bouldin index mainly uses the distance separating the clusters as a metric for evaluating g. The overall execution of the local selection phase of QASSA is described in Algorithm 1.

D. Global Selection Phase

The global selection phase aims at composing locally selected services and determining near-optimal service compositions, i.e., service compositions that: (i) satisfy the global QoS requirements, and (ii) maximise the QoS offered to the user. In our approach, we focus on selecting several alternative service compositions (and not a single composition). Specifically, we aim at selecting many services for each activity in the user task, such that, no matter what is the service finally executed for each activity, the resulting composition satisfies the global QoS requirements imposed by the user on the task.

When dealing with global optimisation problems with multiple objectives and a large number of potential solutions, heuristic algorithms are the only possible choice [10]. We focus on using population-based heuristics (e.g., genetic algorithms (GA), differential evolution (DE) algorithms [11]) to solve QoS-aware service selection under global QoS requirements. This class of algorithms consists in recursively optimising an initial solution using operators such as crossover/mutation. Starting from the fact that our local selection approach is highly selective in the sense that it selects few services having a high QoS level (which reduces considerably the number of services to be investigated), we argue

```
input : A set of activities $T = \{A_1, .., A_z\}$ of size $z$;
        A set of services $S_i = \{s_{i,1}, .., s_{i,m_i}\}$ of size $m_i$ for each activity $A_i$
($i \in \{1, .., z\}$);
        A set of QoS properties $P = \{p_1, .., p_n\}$ of size $n$;
        A set of weights on QoS properties $W = \{w_1, .., w_n\}$ of size $n$;
        A QoS vector $QoS_{s_{i,k}} = \langle q_1, .., q_n \rangle$ for each service $s_{i,k}$ ($k \in \{1, .., m_i\}$).
output: A QoS class for each activity $A_i$
begin
    foreach $A_i \in T$ do
        // Services' clustering
        foreach $p_j \in P$ do
            (Step 1) Apply K-means++ to group the services $S_i$ into $g$
            clusters w.r.t. their values for $p_j$
            K-means++($S_i, g$) $\Rightarrow \{c_{1,j}, .., c_{g,j}\}$;
        end
        // Services' selection
        for $l = g$ downto 1 do
            for $e = n$ downto 1 do
                (Step 2) Build $QL_l$ as the set of $n$ clusters of level $l$ for all
                the QoS properties
                $QL_l = \{c_{l,1}, c_{l,2}, .., c_{l,n}\}$;
                (Step 3) Select a combination $C^e_{|QL_l|}$ of $e$ clusters among
                those of $QL_l$
                foreach $C^e_{|QL_l|}$ do
                    (Step 4) Build QoS class $e$ of the level $l$ as the
                    intersection of the clusters belonging to $C^e_{|QL_l|}$
                    $QC_{l,e} = \{s_{i,k} : s_{i,k} \in \bigcap C^e_{|QL_l|}\}$;
                    (Step 5) Initialise the QoS class to be selected and its
                    quality indicator
                    Result = $\emptyset$;
                    Result_$I_q$ = 0;
                    (Step 6) Select the QoS class with the highest quality
                    indicator
                    if $QC_{l,e} \neq \emptyset$ then
                        $I_q = r \times e \times \sum_{j=1}^{y} w_j$;
                        if $I_q > Result\_I_q$ then
                            Result_$I_q$ = $I_q$;
                            Result = $QC_{l,e}$;
                        end
                    end
                end
            end
        end
        return Result;
    end
end
```

Algorithm 1: The local selection algorithm

that a population-based heuristic can quickly produce near-optimal service compositions.

In accordance with the above, we adopt a population-based approach to solve the global selection phase of QASSA, viz., the Controlled Random Search (CRS) algorithm, which is a population-based global optimisation heuristic like GA and DE. The choice of CRS is motivated by the fact that it is a simple algorithm capable of global optimisation, subject to inequality constraints [12]. CRS initially builds a preliminary service composition by selecting a service (among those resulting from the local selection phase) for each abstract activity in the user task. The global QoS of the composition is then computed with respect to the structure of the composition and QoS aggregation formulae (as detailed in our previous work [2]. If the global QoS meets the user requirements, the composition is then considered as a solution. The global QoS of the composition is then gradually enhanced by replacing the current worst service in it with a better service.

E. *Computational Complexity Analysis*

We analyse the computational complexity of the local and global selection phases of QASSA. Concerning the local selection phase, we do not consider the complexity of computing the Davies-Bouldin index (which is used to decide about the number of clusters, see Section II-C2), since the computation is performed off-line, i.e., after executing QASSA.

As already explained, the local selection phase is performed using the K-means++ algorithm, the complexity of which is of $O(log(K))$ [8] where K denotes the number of clusters. As we cluster services (i.e., execute K-means++) for each QoS property and for all the activities in the user task, the overall complexity of local selection is then of $O(Z.N.log(K))$, where Z and N denote the number of activities in the user task and the number of QoS properties, respectively. Therefore, the local selection phase runs in a linearithmic time.

Concerning the global selection phase, its computational complexity can be determined based on the fact that we proceed similarly to the CRS algorithm [12]. That is, when iteratively checking service compositions, we replace a single service per composition in each iteration. In accordance with this, we first compose Z services (one service per activity) to build the initial service composition, then we iterate on checking the remaining $T - Z$ services (one service per iteration), where T denotes the total number of services associated with all the activities in the user task. Therefore, the total number of compositions to check (i.e., more specifically the number of iterations) is $T - Z + 1$. Additionally, for each composition, we execute $Z.N$ arithmetic instructions to aggregate the N QoS values of the Z services forming the composition. Then, we execute N comparison instructions to determine whether the N QoS values of the composition satisfy the global QoS requirements of the user. The global selection phase runs then in quadratic time of $O((Z.N+N)(T-Z+1))$.

Based on the above results, we state that QASSA executes in quadratic time, thus it reduces considerably the computational complexity of service selection under global QoS requirements, known to be NP-hard [13].

III. Distributing QASSA

The version of QASSA presented in Section II assumes the presence of a centralised and stationary infrastructure supporting QoS-aware service composition. Nevertheless, this infrastructure presents several drawbacks, mainly dealing with scalability, fault-tolerance, and privacy [14]. Additionally, within ubiquitous environments, it is not always possible to assume the support of such an infrastructure. QoS-aware service composition in ubiquitous environments (i.e., more specifically selection algorithms) should rather rely on *ad hoc* topologies with no infrastructure support. For this reason, we present a distributed version of QASSA, which is capable of operating on top of *ad hoc* infrastructures formed of mobile and resource-constrained devices.

Distributed QASSA enables accomplishing service selection as a synergistic interaction between the user device (referred to as requester) and other devices available in the environment (referred to as helpers). As described in Algorithm 2, the main idea of our distributed QASSA is to perform local selection for each activity in the user task using a helper, thus enabling to execute the whole local selection phase using several helpers simultaneously. After that, the requester collects the local selection results and performs the global selection phase on the user device.

The global service selection is difficult to carry out in a distributed way because it requires a global vision of QoS information and the structure of the composition [15]. Additionally, it typically requires a resource-rich device, given the computational complexity of the problem. As detailed in Section II-E, our global service selection algorithm has a low computational complexity; thus it can be carried out using only the resource-constrained device of the requester. The timeliness of our distributed algorithm is further validated by experimental results detailed in the next section.

```
input : A set of activities $T = \{A_1, .., A_z\}$ of size $z$;
        A set of services $S_i = \{s_{i,1}, .., s_{i,m_i}\}$ of size $m_i$ for each activity $A_i$
($i \in \{1, .., z\}$);
        A set of QoS properties $P = \{p_1, .., p_n\}$ of size $n$;
        A set of weights on QoS properties $W = \{w_1, .., w_n\}$ of size $n$;
        A QoS vector $QoS_{s_{i,k}} = \langle q_1, .., q_n \rangle$ for each service $s_{i,k}$ ($k \in \{1, .., m_i\}$).
output: A set of service compositions.
begin
    (Step 1) Splitting the user request into a set E of elementary requests
    foreach $A_i \in T$ do
        $e_i = (A_i, S_i, P, W)$;
        E = E ∪ $e_i$;
    end
    (Step 2) Broadcasting help message and getting helpers
    broadcast (help message);
    foreach device d favorably replying to the help message do
        helpers = helpers ∪ $d$;
    end
    while E ≠ ∅ do
        (Step 3) Scheduling elementary requests to helpers
        foreach $e_i \in$ E do
            $e_i = d$        ($d \in$ helpers);
        end
        (Step 4) Fulfilling the local selection phase
        while Timeout Session do
            Helper Side : execute local selection (Algorithm 1) given
            $e_i$ as input;
        end
        (Step 5) Getting the results of elementary requests
        foreach $e_i \in$ E do
            if result ($e_i$) ≠ null then
                Selected services for $A_i$ = result ($e_i$);
            end
            else Break;
            E = E \{$e_i$};
        end
    end
    (Step 6) Fulfilling the global selection phase
    execute the CRS algorithm given the selected services for $T$;
end
```

Algorithm 2: Overview of distributed QASSA from the *requester* point of view. The coloured box concerns the execution at the *helper* side.

IV. Experimental Evaluation

We conducted a set of experiments to assess the centralised and distributed versions of QASSA. Table I describes the experimental set up used in our experiments. It worth noting that we use basic setup (with limited computational and memory resources) that can be readily assumed in the context of ubiquitous environments. For the evaluation of QASSA, we are interested in two metrics:

- *Execution time*: It measures the timeliness of QASSA with respect to the size of the selection problem in terms of the number of activities and the number of candidate services per activity.

Centralized algorithm	Distributed algorithm
- Machine: Dell - Processor: AMD Athlon 1.80GHz - RAM: 1.8 GB - OS: Windows XP - Programming language: J2SE 1.6	- Machine: HTC Desire - Processor: Qualcomm QSBD8250 1GHz - RAM: 576 Mo - OS: Android 2.2 (Froyo) - Programming language: Android SDK 2.2 (based on J2SE 1.5)

TABLE I
Experimental set up

- *Optimality*: It measures how optimal is the QoS utility provided by QASSA. This is determined by the ratio of the QoS utility resulting from QASSA over the optimal QoS utility given by a brute-force algorithm. The optimality metric is then given by the following formula:

$$Optimality = \mathrm{F}/\mathrm{F}_{opt} \tag{3}$$

  where F is the QoS utility given by our heuristic algorithm, and F_{opt} is the optimal QoS utility given by the IBM ILOG CPLEX Optimiser[1].

For the purpose of our experiments, we developed a *Composition Generator*, which randomly generates service compositions used for experimenting with QASSA. The Composition Generator takes as parameters the number of activities a and the number of candidate services per activity k, and it proceeds through two steps: (i) yielding an abstract service composition which comprehends a activities structured with respect to randomly chosen composition patterns, (ii) binding k concrete services to each activity in the composition. QoS values associated with these services are acquired from the QWS dataset available online[2]. This dataset consists of 5000 real Web services, each with a set of 9 QoS properties measured using commercial benchmark tools [16]. Further details about the implementation of QASSA are given in [17].

A. Performance of QASSA (the centralised version)

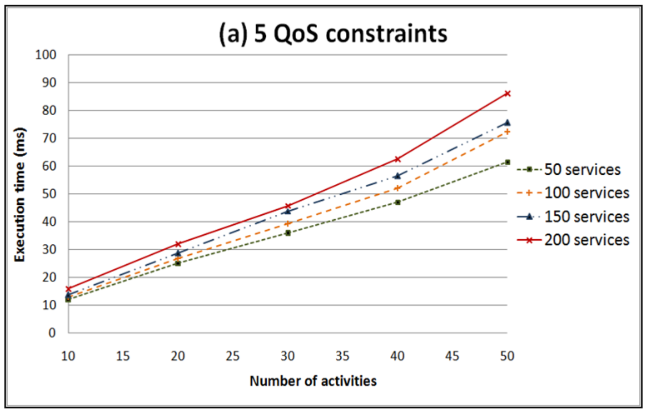
In this section, we present the experimental evaluation of the centralised version of QASSA (using the experimental setup detailed in the left column of Table I). For the sake of precision, we execute each experiment 20 times and we calculate the mean value of the obtained results.

Figure 2 (a) depicts the execution time of QASSA with respect to the number of services per activity. We fix the number of QoS constraints to 5, vary the number of activities between 10 and 50, and vary the number of services per activity between 50 and 200. The obtained results show that the execution time of our algorithm increases (up to 89ms) along with the number of services, which is an expected result.

Figure 2 (b) depicts the execution time of QASSA with respect to the number of QoS constraints. We fix the number of services per activity to 200 and vary QoS constraints between 2 and 5. The obtained results show that the execution time of our algorithm increases (up to 89ms) along with the number of QoS constraints, which is also an expected result, i.e., a higher number of QoS constraints requires more computational effort, hence a longer execution time.

Both figures show that the execution time of our algorithm increases almost linearly along with the number of activities

[1]http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/about/?S_CMP=rnav
[2]http://www.uoguelph.ca/~ qmahmoud/qws/index.html



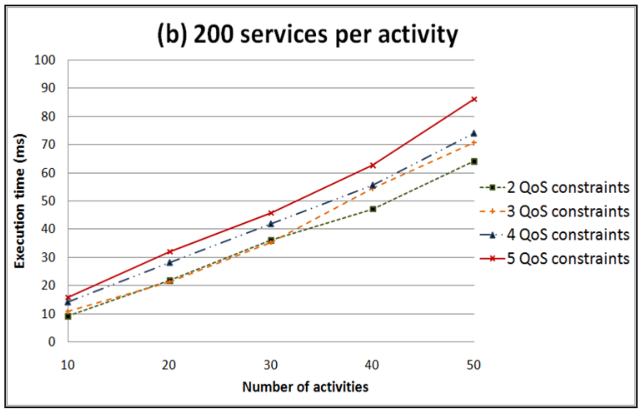


Fig. 2. Execution time while varying (a) the number of services per activity, and (b) the number of QoS constraints

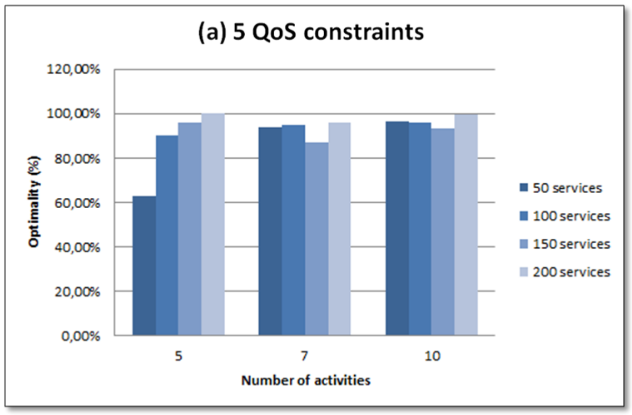
in the composition. In general, our algorithm executes in a timely manner (i.e., less than 0.09s) with respect to spontaneous interaction with users aimed at by ubiquitous computing. Indeed, guidelines for response time in interactive applications specify that 1s is the limit to keep the user's flow of thought seamless [18].

To have a more accurate idea about the efficiency of QASSA in terms of timeliness, we compare the above obtained results to those published in [13], which is a recent work that also combines local and global selection techniques for service selection under global QoS constraints. The authors study the same dataset (i.e., QWS), as well as the same parameters as in our experiments. Although that the considered work uses an experimental setup (a HP ProLiant DL380 G3 machine with 2 Intel Xeon 2.80GHz processors and 6 GB RAM) that is more powerful than the setup used for evaluating QASSA, both works have roughly the same execution time (between ≈10 ms and ≈90 ms).

Concerning the optimality of QASSA, we measure it while varying the number of activities between 5 and 10. Figure 3 (a) depicts the optimality of QASSA while fixing the number of QoS constraints to 5, varying the number of activities between 5 and 10 and varying the number of services per activity between 50 and 200. It shows that the optimality of QASSA is generally more than 90%, and it can reach 100%. However, for the specific case of 5 activities and 50 services per activity, the optimality decreases to 60%, which can be explained by the fact that when the number of services decreases, the probability to find services with a satisfactory value for all QoS properties decreases also, hence yielding a low optimality.

Additionally, we measure the optimality of QASSA while fixing the number of services to 200, varying the number of activities between 5 and 10 and varying the number of QoS constraints between 2 and 5. Figure 3 (b) shows that the optimality of our algorithm is generally satisfactory (more than 90%) and it can reach 100%. Overall, both figures show that the optimality of our algorithm varies between 90% and 100% independently from the number of services and the number of QoS constraints, except for low service populations.

Comparing the optimality of QASSA to the optimality of [13], for the same configuration (as both works use different ranges of values), e.g., a user task of 10 activities each having 200 candidate services to be selected under 5 global QoS constraints, both works produce nearly the same optimality (≈ 100%).



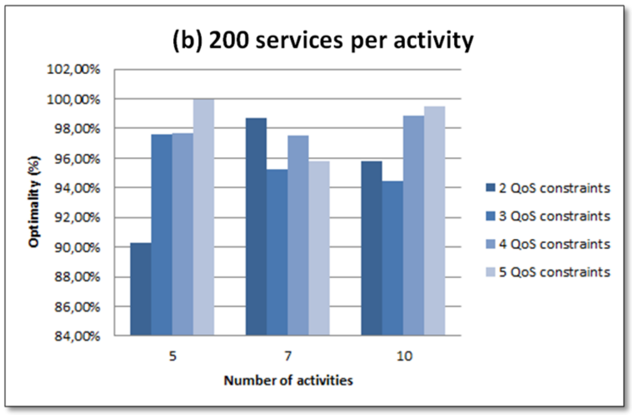
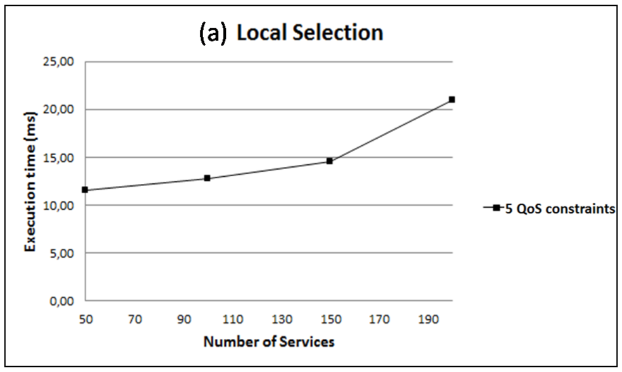


Fig. 3. Optimality measurements while (a) varying the number of services, and (b) the number of QoS constraints

B. Performance of QASSA (the distributed version)

We now evaluate the distributed version of QASSA using the experimental setup detailed in the right column of Table I. The distributed design of QASSA changes two main factors compared with the centralised version, notably: (i) the communication cost between the devices participating in fulfilling the user task, and (ii) the hardware setup underpinning the execution of the algorithm. Both features do not impact the optimality of QASSA, thus the following experiments focus only on the execution time metric. Additionally, we assume that the communication cost is negligible compared to the overall execution time of the algorithm (further details about the network delay can be found in, e.g., [19]). Thus, the execution time presented in these experiments concerns only



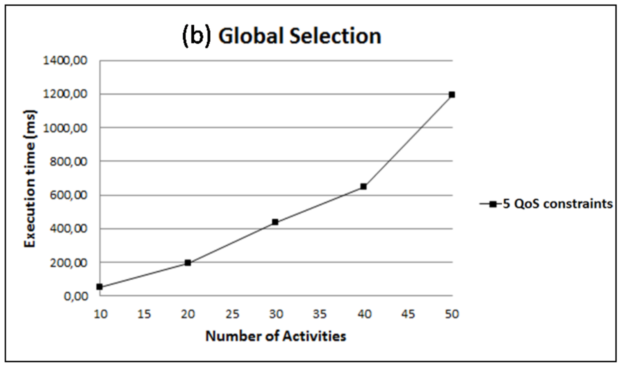


Fig. 4. Execution time of the (a) local selection and (b) global selection of our distributed QoS-aware service selection algorithm

the local and global selection phases of the distributed version of QASSA (see Figure 4). For the local selection, the execution time is measured for only one activity (indeed, each helper device processes in parallel local selection for a single activity in the user task). We fix the number of QoS constraints to 5 and vary the number of services between 50 and 200. Whereas for the global selection, we fix the number of QoS constraints to 5, the number of services to 200, and we vary the number of activities in the user task between 10 and 50.

Despite the relatively limited hardware resources used in these experiments, QASSA shows satisfactory timeliness with respect to on-the-fly service composition in ubiquitous environments. Indeed, the local selection is executed in at most 25 ms, whereas the global selection is executed in at most 1.2 s, thus the overall algorithm can be accomplished in less than 1.5s, depending on the size of the user task and the number of services per activity [18].

V. Related Work

Surveying QoS-aware service selection algorithms represents a broad topic. In this section, we focus on algorithms in line with the recent trend of combining local and global selection techniques. A first research effort in this direction is proposed by Alrifai et al. [20]. The authors present a selection algorithm that starts from the global level and resolves the selection problem at the local level. Indeed, they proceed by decomposing global QoS constraints (i.e., imposed by the user on the whole composition) into a set of local constraints (i.e., for individual sub-tasks, parts of the whole composition). To do so, they use MILP techniques to find the best decomposition of global QoS constraints. The main drawback of this approach is that it relies on a greedy method for the decomposition of QoS constraints, which produces strict local QoS constraints that discard a lot of service candidates. In a more recent work [13], the authors attempt to enhance their approach by relaxing the local QoS constraints as much as possible while not violating the global constraints. While their recent approach may improve the obtained results, conceptually it does not resolve the problem of discriminating potential service candidates.

To cope with this issue, the same authors present another approach [21] combining local and global selection techniques, but in another way. The authors start by the local selection phase. They use two techniques to reduce the number of services investigated for each activity in the user task. First, they use the *skyline* concept [22] as a technique to determine the most interesting services in terms of QoS. Once skyline services are determined, the authors cluster them into several clusters using K-means, and then they select a representative service for each cluster. At the global level, the authors compose the representative services selected at the local level, and check whether the composition meets global QoS requirements using MILP. This approach also presents several drawbacks. Concerning the algorithm itself, the authors claim finding the optimal service composition, because they assume that skyline services are the best services in terms of QoS, which is not true. Indeed, a skyline service is a service that has the highest (i.e., the best) value for one or more QoS properties, whereas for the remaining QoS properties it may have very low values. Regarding this definition, it is possible that a non-skyline service with high values (and not the highest) for all QoS properties yields a higher overall QoS than a skyline service. Concerning the performance of the algorithm, during the local selection phase the authors execute K-means $Z.(T/2)$ times (where Z is the number of activities in the user task, and T is the number of service candidates investigated for a given activity), which represents a high number of iterations, especially when it deals with a large number of service candidates. In our approach, we execute K-means++ (which already outperforms K-means) $Z.N$ times where N is the number of QoS properties. The complexity of our local selection phase is then reduced compared to [21], since the number of QoS properties is always limited compared to the number of service candidates. Additionally, at the global selection phase, the authors execute MILP iteratively until a near-optimal composition is found. In each iteration, the set of representative services with the highest QoS utilities is investigated. This approach may end up executing MILP α times, where α is the number of representative services, which means also a high number of iterations when it deals with a large number of representative services.

Another approach combining local and global selection techniques is presented by [15]. Similar to [20], the authors decompose global QoS constraints into local constraints using MILP. Based on the local QoS constraints, they select services for each activity in the user task. Then, they compose the locally selected services and check whether the composition meets global QoS constraints, using MILP again. The main advantage of this approach is that it executes local selection in a distributed way similarly to our approach. However, they decompose the global QoS constraint imposed on a given QoS property into the average values of that property associated with the services of each activity, which is not accurate and may discriminate a number of service candidates. A similar approach is presented by Jin et al. [23]. The authors decompose global QoS constraints into local constraints using MILP, then they perform local selection. The main shortcoming of this approach is that it does not guarantee meeting global QoS requirements.

An interesting approach is presented by Liu et al. [24]. The authors propose a QoS-aware service selection algorithm which also combines local and global selection techniques. They use the *convex hull* concept [25] as a local selection technique. At the global level, the authors randomly establish an initial composition, and they try to enhance it using services selected by the convex hull. The main drawback of this approach is that it closely depends on the initial composition.

VI. Conclusion

This paper introduces QASSA, a QoS-aware service selection algorithm. Conceptually, QASSA defines service selection under global QoS requirements as a set-based bi-level optimisation problem, which represents a mathematical model for the problem. Additionally, QASSA handles the complexity of the problem by combining local and global selection techniques, and solves the local selection problem using service clustering in a novel way (i.e., intersection of one-dimensional service clusters). Moreover, QASSA considers jointly: (i) centralised and distributed design fashions, and (ii) support for adaptation at run-time by selecting several alternative service compositions instead of only one. In practise, QASSA shows satisfactory timeliness and optimality, hence representing an efficient mean for building complex user tasks in ubiquitous environments.

References

[1] E. Zitzler, L. Thiele, and J. Bader, "On Set-based Multiobjective Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 1, pp. 58–79, 2010.

[2] N. Ben Mabrouk, S. Beauche, E. Kuznetsova, N. Georgantas, and V. Issarny, "QoS-aware Service Composition in Dynamic Service Oriented Environments," in *Middleware'09: Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*. Springer Verlag New York, Inc., 2009, pp. 1–20.

[3] J. Bard, *Practical Bilevel Optimization: Algorithms and Applications*. USA: Springer, 1998, vol. 30.

[4] W. Jiang, S. Hu, and Z. Liu, "Top K Query for QoS-Aware Automatic Service Composition," *IEEE Transactions on Services Computing*, vol. 99, no. PrePrints, p. 1, 2013.

[5] H. Al-Helal and R. Gamble, "Introducing Replaceability into Web Service Composition," *IEEE Transactions on Services Computing*, vol. 99, no. PrePrints, p. 1, 2013.

[6] H. Ma, F. Bastani, I.-L. Yen, and H. Mei, "QoS-Driven Service Composition with Reconfigurable Services," *IEEE Transactions on Services Computing*, vol. 6, no. 1, pp. 20–34, 2013.

[7] S. Verel, A. Liefooghe, and C. Dhaenens, "Set-based Multiobjective Fitness Landscapes: A Preliminary Study," in *GECCO'11: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: ACM, 2011, pp. 769–776.

[8] D. Arthur and S. Vassilvitskii, "K-means++: The Advantages of Careful Seeding," in *SODA'07: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[9] D. L. Davies and D. W. Bouldin, "A Cluster Separation Measure," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 1, pp. 224–227, 1979.

[10] A. Grosso, M. Locatelli, and F. Schoen, "A Population-based Approach for Hard Global Optimization Problems based on Dissimilarity Measures." *Math. Program.*, vol. 110, no. 2, pp. 373–404, 2007.

[11] M. M. Ali and A. Törn, "Population Set-based Global Optimization Algorithms: Some Modifications and Numerical Studies," *Computers and Operations Research*, vol. 31, no. 10, pp. 1703–1725, 2004.

[12] W. Price, "Global optimization by controlled random search," *Journal of Optimization Theory and Applications*, vol. 40, no. 3, pp. 333–348, 1983.

[13] M. Alrifai, T. Risse, and W. Nejdl, "A Hybrid Approach for Efficient Web Service Composition with End-to-end QoS Constraints," *ACM Transactions on the Web*, vol. 6, no. 2, pp. 7:1–7:31, Jun. 2012.

[14] H. Fernandez, C. Tedeschi, and T. Priol, "A Chemistry-Inspired Workflow Management System for a Decentralized Workflow Execution," *IEEE Transactions on Services Computing*, vol. 99, no. PrePrints, p. 1, 2013.

[15] J. Li, Y. Zhao, M. Liu, H. Sun, and D. Ma, "An Adaptive Heuristic Approach for Distributed QoS-based Service Composition," *IEEE Symposium on Computers and Communications*, vol. 0, pp. 687–694, 2010.

[16] E. Al-Masri and Q. H. Mahmoud, "Discovering the Best Web Service," in *WWW'07: Proceedings of the 16th International Conference on World Wide Web*, 2007, pp. 1257–1258.

[17] N. Ben Mabrouk, "Qos-aware service-oriented middleware for pervasive environments," Ph.D. dissertation, Université Pierre et Marie Curie - Paris VI, April 2012, uRL: http://hal.inria.fr/tel-00786466.

[18] J. Nielsen, *Usability Engineering*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[19] A. Klein, I. Fuyuki, and S. Honiden, "SanGA: A Self-Adaptive Network-Aware Approach to Service Composition," *IEEE Transactions on Services Computing*, vol. 99, no. PrePrints, p. 1, 2013.

[20] M. Alrifai, T. Risse, P. Dolog, and W. Nejdl, "A Scalable Approach for QoS-based Web Service Selection," in *QoSCSOA'08: Proceedings of the 1st International Workshop on Quality-of-Service Concerns in Service Oriented Architectures held in conjunction with ICSOC 2008*, Sydney, December 2008.

[21] M. Alrifai, D. Skoutas, and T. Risse, "Selecting Skyline Services for QoS-based Web Service Composition," in *WWW'10: Proceedings of the 19th International Conference on World Wide Web*. New York, NY, USA: ACM, 2010, pp. 11–20.

[22] S. Börzsönyi, D. Kossmann, and K. Stocker, "The Skyline Operator," in *ICDE'01: Proceedings of the 17th International Conference on Data Engineering*. Washington, DC, USA: IEEE Computer Society, 2001, pp. 421–430.

[23] J. Jin, Y. Zhang, Y. Cao, X. Pu, and J. Li, "ServiceStore: A Peer-to-Peer Framework for QoS-Aware Service Composition," in *Network and Parallel Computing*. Springer Berlin / Heidelberg, 2010, vol. 6289, pp. 190–199.

[24] D. Liu, Z. Shao, C. Yu, and G. Fan, "A Heuristic QoS-Aware Service Selection Approach to Web Service Composition," in *ICIS'09: Proceedings of the 2009 Eighth IEEE/ACIS International Conference on Computer and Information Science*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1184–1189.

[25] M. Mostofa Akbar, M. Sohel Rahman, M. Kaykobad, E. G. Manning, and G. C. Shoja, "Solving the Multidimensional Multiple-choice Knapsack Problem by Constructing Convex Hulls," *Computers and Operations Research*, vol. 33, pp. 1259–1273, May 2006.