A Distributed Agent-Based Approach for Optimal QoS Selection in Web of Object Choreography

Nacera Temglit, Abdelghani Chibani, Karim Djouani, and Mohamed Ahmed Nacer

***Abstract*—The Internet of Things (IoT) refers to the domain of physical/logical objects connected to the Internet that can be accessible anywhere and anytime, giving the access policies. Web objects and services can be seen as distributed and cooperative agents that need to collaborate in order to reach advanced functionalities and also to optimize the overall quality offered to the end users. However, the number of the services that can be supplied through these devices on the Internet will increase more and more. In this context, performing the composition and selection of the best services according to quality-of-service (QoS) criteria become a complex task, especially when dealing with global service selection to satisfy global constraints of users in decentralized environments. For this purpose, we propose a distributed and optimal QoS selection approach based on the multiagent paradigm and the distributed constraint optimization problem (DCOP) formalism. Hence, we revisited a DCOP technique for developing an efficient algorithm: the DPOP4QoS implemented under a real-time Web protocol for IoT communication, the extensible messaging and presence protocol. The proposed algorithm takes into account the specificities of the service composition context and the satisfaction of the global user constraints. This paper also presents a set of experiments conducted under realistic distributed environment to evaluate the performance of the proposed algorithm.**

***Index Terms*—Multiagent systems, optimization, quality of service (QoS), service interoperability and composability, ubiquitous computing, Web services.**

I. Introduction

INTERNET of Things (IoT) is emphasizing a world where small intelligent objects are able to collaborate by sharing data and organizing themselves in groups in order to reach complex objectives. The rapid adoption of this technology within the service-oriented approach (SOA) increases the popularity and usage of devices connected through the IoT. According to Gartner,[1] the number of connected devices is increasing to reach more than 5 million of new devices that can be connected every day. This means that the number of the services that can be supplied through these devices on the internet will increase more and more. In this context, performing the composition and selection of these services according to quality-of-service (QoS) criteria will become a complex task for operators and business application designers. For this reason, service composition applies appropriate techniques to allow the highest QoS to be achieved according to the end-user requirements and priorities [1].

In service-oriented computing, service composition can be modeled and implemented according to one of the two following orchestration architectures: centralized orchestration agent or decentralized orchestration based on the cooperation (called also choreography) of several orchestration agents. In the former, the centralized orchestrator controls the selection and invocation of services participating in the composition and manages execution errors and exceptions. In the latter, we assume that a centralized orchestrator cannot handle the selection and invocation due to several technical reasons. Therefore, the composition is distributed between different agents that are sufficiently intelligent to handle the selection and invocation of the services of all the composition in a peer-to-peer mode.

This work has been undertaken in the European project ITEA2 WoO, which aims to propose a new architecture that connects, through Web protocols and services, objects like sensors, actuators, robots, or any physical or virtual object as well to solve the central problem of peer-to-peer services composition. The objective is to select the most appropriate combination of the operational services from those available in the ubiquitous environment to optimize the entire quality of a composite service expected by the end users. QoS values determine whether a Web service is reliable, trustworthy, efficient, or features/has other nonfunctional aspects required by end users. Some examples of common QoS properties include price, response time, reliability, availability, and also energy level of IoT devices. In the case of robotic and automation services, the complexity of the service selection operation will increase as those objects are characterized by their dynamic aspect such as robots mobility, limited connectivity in some situations, and intensive use of battery to handle actuation. These objects that offer the elementary services involved in a composition need to communicate and collaborate in order to share the levels of their QoS with up-to-date information.

In this paper, a distributed optimization approach is proposed to solve the problem of service selection in the context of service choreography architecture. In the literature, various techniques

Manuscript received March 22, 2016; revised June 12, 2016 and September 24, 2016; accepted November 26, 2016.

N. Temglit is with the National High School for Computer Science (ESI), 16000 Algiers Algeria (e-mail: n_temglit@esi.dz).

A. Chibani is with the Laboratory of Images, Signals and Intelligent Systems, University of Paris-Est Créteil, 94000 Créteil, France (e-mail: chibani@u-pec.fr).

K. Djouani is with the French South African Institute of Technology (FSATI), Tshwane University of Technology, Pretoria 0183, South Africa (e-mail: djouani@ieee.org).

M. A. Nacer is with the LSI Laboratory, University of Science and Technology Houari Boumediene, Bab Ezzouar 16111, Algeria (e-mail: med.anacer@gmail.com).

Digital Object Identifier 10.1109/JSYST.2016.2647281

[1]Gartner Special Reports. [Online]. Available: //www.gartner.com/technology/research/

such as linear programming (LP), constraint optimization problem (CSP) formalism, and heuristic-based methods have been proposed for optimal or near-optimal Web service selection problems in the context of centralized and static environments [2]–[5]. Applying these algorithms in a distributed setting would be difficult and entail large communication overheads. Very few approaches have been proposed to solve the problem of optimal service selection in distributed environments [6]–[8], while most of them focused on a local service selection at each service component of the distributed environment aiming to find near-optimal compositions by avoiding communication between objects. However, the optimality is never achieved especially when the constraints of end users are global. In the present work, we aim to tackle this problem by taking advantage of the multiagent paradigm for implementing an approach to select and orchestrate the execution of services. In other words, the architecture is composed of core agents that are in charge of optimizing the global quality of the composed services and to satisfy the QoS user's constraints by selecting the best candidate services. This can be formalized as a distributed constraint optimization problem (DCOP) [9], [10] seeking to maximize or minimize the QoS attributes, such as minimizing response time and maximizing availability at the same time. To address the tradeoff between different objectives of service quality, we take advantage of the simple additive weighting (SAW) technique and use the utility function method, which allows a unified measurement of multiple objectives.

Moreover, in the context of dynamic service environments, the execution time of the distributed service selection algorithms is heavily constrained, as the service availability is not guaranteed and the variation in QoS parameter values is very frequent. For that, in our architecture of service composition, we suggest to reiterate the selection process during the execution of the composite service to take into account the changes that may occur in the environment. This is why the scalability of the selection algorithm is highly required. To be able to address this issue, we propose in this paper a DCOP-based algorithm, the DPOP4QoS, which allows a QoS selection with very low volume of messages exchanged between agents. The DPOP4QoS algorithm revisits the basic dynamic programming optimization protocol (DPOP) [11] and takes also into account: 1) the satisfaction of global user's constraints that cannot be achieved by local selection methods and 2) the composition structures (sequence, parallel, and loop patterns) are considered as local constraints between agents that decide about how to calculate the utility values.

The rest of this paper is organized as follows. In Section II, we give some basic concepts on service composition and selection as well as a review of some related work of the domain. A global IoT architecture for services composition is discussed in Section III. A DCOP formalization of the service selection problem is given in Section IV. The algorithm proposed to solve the distributed service selection problem is discussed in detail in Section V. Evaluation and experimental results for the proposed algorithm "DPOP4QoS" are presented in Section VI, and then, we conclude and give some perspectives in Section VII.

II. Background and Related Work

In the following, the Web service composition and selection problem is discussed, before reviewing main existing approaches, to the best of our knowledge, proposed in the literature for QoS Web service selection as well as some existing DCOP algorithms.

A. *Web Service Composition and Selection*

The true potential of Web services can only be realized through assembling multiple services into more powerful applications with more sophisticated functionalities; this is called Web service composition. We perceive solving a Web service composition in two main processes: Vertical and Horizontal composition [12], [13]. The Vertical composition is aimed at finding a combination of the abstract Web services (abstract composite service), for fulfilling functional requirement of the end users. An abstract Web service describes a service from the functional point of view without referring to any existing service and is represented by only functional attributes as the required inputs for a given service as well as the expected outputs. In the abstract composite service, the component services are arranged according to different workflow patterns or structures that configure them into a new composite service with value-added functionality. The main forms of workflow structures are sequence, choice, parallel, split, and loop. The Horizontal composition is aimed at finding the best concrete Web service, from among a set of available functionally equivalent Web services according to QoS attributes. This is known also as Web service selection or QoS-aware service composition. The QoS attributes encompass a number of nonfunctional properties such as the service price, reliability, response time, reputation, availability, etc. [1] The overall quality of the composite service is calculated aggregately from the quality of its constituent services.

The two straightforward solutions for the service selection problem are global and local selection. The selection is done locally when the selection of concrete services for each abstract service is done independently from the others using local cost function. It is applied when the user constraints are local. However, the users specify, usually, their expectations at global level, which implies that the selection must satisfy global constraints. This approach aims to solve the problem at the composite service level by evaluating all possible combinations of concrete services to choose the best combination in terms of QoS, while satisfying the global constraints. The problem of global selection is considered as a multicriteria and NP complexity problem; it is, therefore, difficult to find an optimal solution in a reasonable time when the number of abstract component services and concrete candidate services become important. This problem will be the focus of this paper.

For modeling and executing service compositions, two main approaches can be used: the centralized approach that is called service orchestration and the decentralized approach that is called choreography. In orchestration mode, a single execution engine is responsible for the invocation of the elementary services that are part of the composite services, including the enforcement of control flows and message exchanges among component services. This differs from service choreography, which is peer-to-peer collaborative execution that is handled by services or agents that monitor their execution [14].

B. QoS Web Service Selection Approaches

In [15], the authors discuss some works on service selection based on trust and reputation systems that permit QoS requirements to be negotiable with users. Therefore, the major inconvenience of these approaches is the involvement of users to ensure the voting system, which requires cooperation based on trust between users. Other techniques are based on graph theory, LP, genetic algorithms (GAs), Pareto search, CSP formalism, and decision-making techniques. We review below some of the main contribution and limitations of the works that fall into these categories.

The work of Zeng and Benatallah [2] focuses on dynamic and quality-driven selection of services. The authors use mixed LP techniques to find the optimal selection of component services. But, these methods assume linearity of the constraints and the objective function and suffer from poor scalability due to the exponential time complexity of the applied search algorithms. In [16]–[22], the proposed methods are based on GAs to solve the selection problem in a reasonable time. However, as users have often to fix *a priori* a constant number of iterations or a time deadline to stop the GA, this does not give any guarantee about the quality of the generated solution. Therefore, the GA offers a better scalability but deemed nonuseful for selecting the optimal composition plan. In [5], Lécué proposes a CSP-based formalism to solve the QoS selection problem. The composition-driven CSP is solved by adapting a stochastic search method, which sacrifices completeness for speed and scalability. The author uses for this purpose the hill climbing algorithm. Another service composition approach based on the Pareto set model is recently proposed in [23]. This work focuses on finding the Pareto set of optimal solutions (compositions) by pruning candidate services and workflows that are not promising ones for optimal compositions. The authors use a QoS-based dominance relationships for this purpose and propose a parallel algorithm to improve efficiency on the composition space size and operation time. However, the approach aims to find a set of feasible solutions instead of a unique solution for the problem. In [24], the authors propose a real-time QoS-aware selection using a multicriteria decision-making technique (analysis hierarchy process—AHP). The AHP is used to weigh user preferences and then uses a cost-benefit ratio to rank service choices of two general cloud services: storage and compute cloud services. However, it is difficult to generalize this idea for more than two services especially with various composition patterns. In [25], the authors propose an optimal Cross-Cloud service composition by performing a multilevel iteration process to mine the optimal or most optimal (trusted) subcompositions at different optimization levels. The nth level includes disjoint groups of two composition subgroups of the $(n-1)$th level (starting by groups of two composition tasks). A threshold is calculated to sift out useless subcompositions at each iteration level (the subcompositions violating the user constraints). The general idea is interesting, but this work considers only the sequential composition pattern and supposes that the top subcompositions at each level have a higher possibility to meet QoS constraints; therefore, the global optimality is still not guaranteed.

There has been very few works in the area of distributed QoS selection. In [26], the authors address the problem of satisfying end-to-end user constraints in a distributed environment by proposing a hybrid approach. The global constraints are decomposed into local constraints using mixed integer programming such that the satisfaction of local constraints at each domain guarantees the satisfaction of global constraints. However, the method of extracting QoS levels from the QoS information of service candidates is greedy (does not deal with dependencies between QoS dimensions); this leads to very restrictive decomposition of the global constraints to local constraints. The authors of [27] and [28] base their approaches on a decomposition of the global objective on the local domains on the Cloud and use a GA to find the best composition. Zheng *et al.*'s work [29] adopted a greedy strategy-based solutions by using the CloudRank algorithm for QoS ranking on the cloud; however, this can only produce locally approximate solutions. In [8], two versions of a Web service selection algorithm are proposed: a centralized version and a decentralized one; so that it can be executed on top of centralized and decentralized infrastructures. In the two versions, the authors combine local and global selection techniques. The local selection aims at selecting services with the highest QoS for each activity in the user task using service clustering techniques, notably the K-means algorithm. The global selection aims at selecting near-optimal compositions of services resulting from the local selection by means of a GA.

C. DCOP Algorithms

The DCOP has been viewed as a powerful paradigm for solving combinatorial problems arising in distributed multiagent environments for the purpose of optimization [9]. DCOP techniques build upon the class of distributed constraint satisfaction problem by finding a complete assignment to the decision variables that not only satisfies the problem constraints but also optimizes the relevant objective(s) [30]. Well-known algorithms used to solve DCOP-like problems include SynchB&B [31], ADOPT [10], DPOP [11], OptAPO [32], and distributed stochastic search algorithms [33]. As for a complete algorithm, an optimum solution is guaranteed, despite the extended computing time. The existing algorithms were evaluated for different application use cases or domain areas, as, for instance, the graph coloring, the meeting scheduling, or the sensor network problems. In [34], the authors provide a good evaluation of ADOPT, OptAPO, and DPOP in terms of the number of cycles, the size/quantity of messages, and the running time for the problem of traffic light synchronization. DPOP is linear in terms of the number of agents. Thus, the DPOP algorithm performed better in terms of the number of exchanged messages, the execution time, and the number of cycles. However, the same cannot be said about the size of the messages. ADOPT needs much more time than the others due to the exponential number of messages. OptAPO is a good compromise in terms of execution time and size of messages under partially distributed systems. Therefore, DPOP seems to be more suited for real-world applications requiring strong limitation in communications.

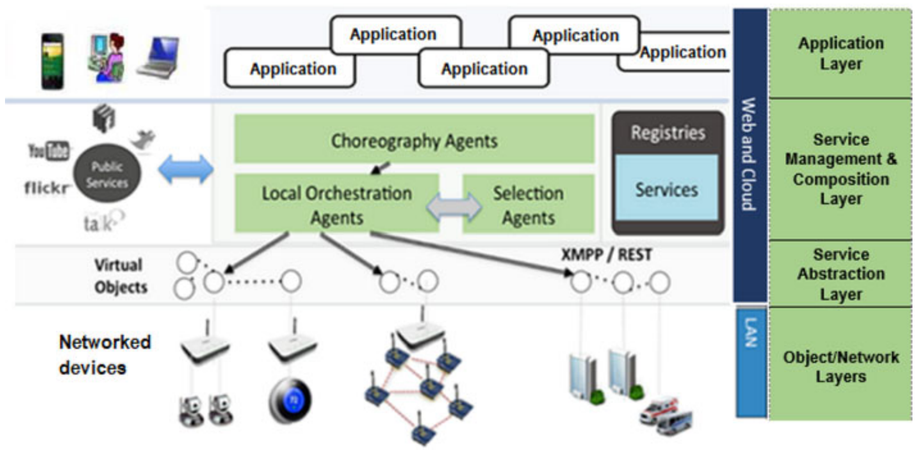

Fig. 1. Architecture of UbiSCo.

III. Architecture of IoT Service Composition

In order to be able to fully exploit the new opportunities of the IoT vision, a scalable and flexible architecture is needed. Nevertheless, the ever-increasing number of the proposed architectures has not yet converged to a reference model [35]–[37]. From the pool of the proposed models, the basic IoT model is a three-layer architecture [38], [39], consisting of the application, network, and object layers. Ubiquitous service composition (UbiSCo) architecture (depicted in Fig. 1) is based on the basic IoT model, to which we add some abstractions in order to integrate IoT into the Web by making them accessible in a SOA way. Indeed, the service abstraction layer and the service composition and management layer are added between the top layer (application) and the two bottom layers (object/network layers). In the service abstraction layer, all functionalities offered by networked devices are abstracted by services using enabled technologies such as REST [40] and extensible messaging and presence protocol (XMPP) [41]. The service composition and management layer is the most critical layer in our work that is responsible for critical functions such as service discovery, composition, execution, and monitoring of services. The service composition and execution are conducted in decentralized mode, where several distributed agents communicate and collaborate in order to handle the selection and invocation of the services in a peer-to-peer fashion [14]. This differs from the centralized orchestration, where a single execution engine is responsible for the selection and invocation of the elementary services. For service composition and execution, we denote three kinds of agents: choreography agents, local orchestration agents, and selection agents. The local orchestration agents ensure the execution of a localized area of services. The choreography agents collaborate to ensure the decentralized execution of the composite service under the specified constraints and optimize the solution. The service selection agents interact using a distributed optimal selection algorithm in order to select the best concrete services among the available ones in their respective domain. The selected service descriptions are sent back to the orchestration agents for execution. The selection process can be reiterated during the execution of the composite service in order to take into account any changes that could occur in the ubiquitous environment, such as variation in the QoS parameters and service unavailability.

All the agents communicate in an adhoc and asynchronous way by using point-to-point and publish subscribe messaging middleware. The point-to-point is used to communicate message to specific agent, whereas publish subscribe is to feed agents with up-to-date notifications. The proposed architecture is also cloud compatible in the sense that some available services could be hosted on the cloud, so as to have a set of clouds with the access grant policies for each cloud.

IV. Distributed Service Selection Approach

In our architecture, the agents need to collaborate and negotiate their quality level in order to select the best concrete services among several similar services according to QoS end-user requirements. The DCOP is an appropriate formalism to manage this interaction. In this section, we will see how the distributed Web service selection can be formalized as a DCOP and what are the specificities to be considered.

A. DCOP Formalism Review

Within the DCOP framework, several agents have to communicate in order to converge to an optimal solution, i.e., the best value of their respective variables leading to the minimal (optimal) value of the global cost. Formally, the DCOP is defined by the 6-uplet (X, D, C, A, ψ, ϕ) with:

1) X the set of distributed variables $\{x_1, x_2, \ldots, x_n\}$;
2) A the agents set $\{A_1, A_2, \ldots, A_k\}$;
3) $\psi : X \rightarrow A$ the mapping function that maps each variable of X to an agent of A;
4) $D = \{D_1, D_2, \ldots, D_n\}$ a set of finite sets, where D_i is the domain of the variable x_i;
5) $C = \{c_{ij} : D_i \times D_j \rightarrow \mathrm{R}^+$, where $i, j = 1, \ldots, n$ and $i \neq j\}$ a set of constraints, corresponding to the local cost function for each couple of variables x_i and x_j.

Only the agent owner of a variable knows the variable domain and has the control on its value. $\phi(Af)$ is the objective function to optimize (maximize or minimize) and Af is the assignment function, which associate to each variable x_i a value $d_i \in D_i$.

TABLE I
QoS Parameters or Metrics

Parameters class	Designation & purpose	
Dynamic Parameters	Energy level (EL)	Actual Battery's energy level of the Object hosting the service
	Availability (AV)	Represents the accessibility of the service. It is the number of successful invocations over the total number of invocations
	Response Time (RT)	Represents the time needed by a given service to respond to the request
	Reliability (RE)	represents the services capacity at ensuring reliable message delivery.
Static Parameters	Price (Price)	the amount of money to pay for each service request
	Security (Secur)	Represents the security level ensured by a service (authentication, encryption, etc.)

$\phi(A)$ is the summation of local costs for each couple of variables sharing one constraint

$$\phi(Af) = \sum_{x_i, x_j \in X} c_{ij}(Af(x_i), Af(x_j)). \quad (1)$$

DCOPs are commonly visualized as constraint graphs, whose vertices are the agents and whose edges are the constraints. The majority of the DCOP/DisCSP algorithms transform this graph into a depth first search (DFS) tree in order to allow a local communication between agents: each agent communicates only with its neighbors (children, parent).

B. QoS Modeling

A QoS model must capture the descriptions of the important aspects of an ubiquitous service and take into account the specificities of each QoS parameter. Hence, we propose a model that allows for specifying generic QoS attributes like Response time, availability, reliability, and price of the service as well as domain-specific QoS attributes, e.g., the Energy Level that indicates the actual Battery's energy level of the ubiquitous object hosting the service. We add to our model the Security Level, which represents the ability of the service to provide appropriate security mechanisms like encryption and authentication. These QoS attributes are classified into two categories (see Table I) [42].

1) Static parameters (SQP for static quality parameters): SQP values are generally known at the deployment or advertisement time and are usually not updated during the execution, for example, the Price and the Security Level of a service.
2) Dynamic parameters (DQP for dynamic quality parameters): DQP parameters represent the variable characteristics of a given service such as the Response time, Energy level, Availability, and Reliability. DQP attributes are determined at the invocation time and their values are provided by a monitoring process.

The QoS attributes are also qualified as negative or positive attributes. In negative attributes, the lower the value, the better the quality like Response time and Cost. In positive attributes, the higher the value, the better the quality like Availability and Reliability. Therefore, the QoS of a concrete service cs_i is denoted by a vector of the six QoS attributes that can be represented as follows: $\mathrm{QoWS}(\mathrm{cs}_i) = (\mathrm{Price}(\mathrm{cs}_i), \mathrm{Secur}(\mathrm{cs}_i), \mathrm{AV}(\mathrm{cs}_i), \mathrm{EL}(\mathrm{cs}_i), \mathrm{RT}(\mathrm{cs}_i), \mathrm{RE}(\mathrm{cs}_i))$.

TABLE II
Aggregation of QoS Values Based on the Pattern: S, P, L

	S (Sequential)	P (Parallel)	L (Loop N)
Price	Price(a)+Price(b)	Price(a)+Price(b)	N*Price(a)
Secur	min(Secur(a),Secur(b))	min(Secur(a),Secur(b))	Secur(a)
EL	(EL(a)+EL(b))/2	min(EL(a),EL(b))	$\sum_{i=0}^{N} \mathrm{EL}_i$
RE	RE(a)*RE(b)	RE(a)*RE(b)/2	$\prod_{i=0}^{N} \mathrm{RE}_i$
RT	RT(a)+RT(b)	max(RT(a),RT(b))	$\sum_{i=0}^{N} \mathrm{RT}_i$
AV	AV(a)*AV(b)	AV(a)*AV(b)	$\prod_{i=0}^{N} \mathrm{AV}_i$

The QoS of the composite service is calculated based on the aggregation of all the QoS vectors representing the QoS of the concrete services involved in the composition. This aggregation depends on the structure of the plan (composition patterns) and the type of parameter. In next section, we will see how to calculate QoS values of a composite Web service according to the candidate QoS values and how we can sort the best combination of concrete services ensuring the optimal global quality according to the DCOP formalism.

C. Problem Formulation

An abstract composite service consists of a set of n services denoted as $\mathrm{AS} = \{\mathrm{as}_1, \mathrm{as}_2, ..., \mathrm{as}_n\}$. For each service $\mathrm{as}_i \in \mathrm{AS}$, there are m candidate services proper to implement it, which are represented by $\mathrm{CS}_i = \{\mathrm{cs}_1^{\mathrm{as}i}, \mathrm{cs}_2^{\mathrm{as}i}, ..., \mathrm{cs}_m^{\mathrm{as}i}\}$. The problem is to select distributively one concrete service $\mathrm{cs}_j^{\mathrm{as}i}$ for each as_i of AS that optimize the global quality of the whole composite service. This is formulated as a DCOP as follows.

1) Each abstract service as_i of the abstract composite plan is associated to a variable x_i controlled (owned) by one agent A_i in the DCOP context.
2) The domain of values of each abstract service as_i is the set of concrete services of its class: $D(\mathrm{as}_i) = \{\mathrm{cs}_j^{\mathrm{as}i}/1 \leqslant j \leqslant m\} = \mathrm{CS}_i$.
3) The value of a concrete service cs_i is the vector of its QoS attributes values. Therefore, the agents will exchange vectors of real values and not single values.
4) The graph of constraints is the composition plan that describes the dependence between the different services involved in the composition according to the three basic composition patterns considered in our work: sequential, parallel, and loop patterns (see Table II).
5) The local cost c_{ij} between two given services is computed based on the type of QoS attributes and on the pattern of composition connecting the considered services (see Table II).

Table II shows the aggregate function of QoS values according to three basic composition patterns: sequential, parallel, and loop. For QoS computation, the parallel composition pattern is the AND Join workflow pattern because the quality of two parallel services is calculated at the synchronized point. The sequence pattern includes the SEQUENCE and AND/OR split workflow patterns, all the services being in split with a common service (S) are considered to be in sequence with this service (the branches of a split pattern may or may not be synchronized).

The local cost $c_{a,b}$ between two concrete services a and b is computed using the weighted sum of the aggregate function values on all QoS attributes according to the SAW technique used for multiobjective problems

$$c_{a,b} = \sum_{(k=1)}^{L} \text{Agg}(q_k(a), q_k(b)) * w_k \tag{2}$$

where $\sum_{i=1}^{L} w_i = 1$, L is the number of QoS attributes, and Agg denotes the used aggregation operator, as described in Table II .

The global cost function $\phi(Af)$ is always the sum of the local costs $c_{a,b}$ for each couple of related variables in the composition plan.

We also need to extend the DCOP algorithm to support satisfaction of global constraints on attributes specified by the user; these are bounds on the QoS attributes: upper bounds for the negative attributes and lower bounds for the positive attributes. For example, the user may expect that the entire service availability shall never be under 60% and the price should not exceed 80 dollars.

Generally, let V_r denote the rth QoS attribute value obtained for a concrete composite service (a feasible solution); the QoS Constraint vector of L' QoS attributes ($L' \leqslant L$) is expressed as $V = (V_1, \ldots, V_{L'})$. bound$_r$ denotes the rth QoS bound on attribute r. The QoS bound vector on L' QoS attributes is expressed as bound $= (\text{bound}_1, \ldots, \text{bound}_{L'})$. A Boolean function Constraint (V_r, bound$_r$, Op$_r$) is defined to verify whether the rth constraint specified by the user (bound$_r$) is satisfied by V_r (according to Op_r relation); the operator op$_r$ can be $>, <, \leqslant, \geqslant, or =$. If the constraint is satisfied, the value of the Boolean function Constraint is true, otherwise false. V_r is obtained as

$$V_r = \sum_{a,b \in \text{CS}} \text{Agg}(q_r(a), q_r(b)) \tag{3}$$

with $q_r(a)$ and $q_r(b)$ denote the given rth QoS attribute values of the concrete services a and b, respectively.

The user may have also preferences on the QoS attribute by specifying weight for each QoS attribute according to its importance (from the user point of view). Therefore, the user preferences are denoted by a vector $W = (w_1, w_2, \ldots, w_L)$ such that $\sum_{i=1}^{L} w_i = 1$. Preferences are taken into account when computing local and global cost.

Therefore, solving the problem of service selection is to find a set of concrete services CS $= \{\text{cs}_1, \text{cs}_2, \ldots, \text{cs}_n\}$ where cs$_i$ is the concrete service selected for the abstract service as_i such as the following.

1) The global cost CostG of CS, defined by the function ϕ in formula (1), is minimized.

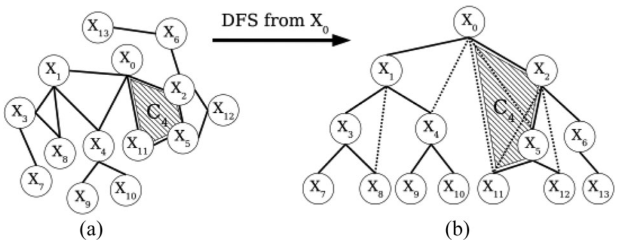

Fig. 2. Example of a DFS tree [43]. (a) Simple graph. (b) Depth-first traversal from X_0.

2) The global quality of CS on each QoS attribute (V_r) must satisfy the user's constraints (bound$_r$)

$$\bigwedge_{r=1}^{L'} \text{Constraint}(V_r, \text{bound}_r, \text{Op}_r) = \text{true}. \tag{4}$$

To resolve our problem, we have revisited the DPOP algorithm, in which only a linear number of messages are exchanged between agents. This is important in distributed settings because sending a large number of small messages (like SynchB&B and ADOPT algorithms do) typically entails large communication overheads especially if the algorithm has to be reiterated several times. We defined new aggregation operators to take into account the composition patterns (when calculating the overall cost) and also the satisfaction of end-user constraints. This will be detailed in the two next sections.

V. DPOP-Based Algorithm for Distributed and Optimal Web Service Selection

The DPOP is a DCOP algorithm having the important advantage that it generates only a linear number of messages. To use the DPOP, the graph of constraints must be transformed into a DFS tree. The DFS structure is potentially better than traditional search on linear variable orderings. The reason is that when performed on a DFS structure, search can be done in parallel on distinct branches of the tree.

A. DFS Concepts

A DFS arrangement of a graph G is a rooted tree with the same nodes and edges as G and the property that adjacent nodes from the original graph fall in the same branch of the tree [43].

In DFS structures, only neighboring agents can communicate directly by sending and receiving messages (local communication). In DPOP, two types of messages are exchanged: UTIL messages sent by agents to their parent (bottom-up messages) and VALUE messages sent by agents to their children (top-down messages).

For each node X_i in a given DFS Tree T of a graph G, we have the following.

1) The children C_i/parent P_i of node X_i are the descendants/ancestor of X_i, which are connected to X_i in T (e.g., from Fig. 2(b), $P_4 = \{X_1\}, C_1 = \{X_3, X_4\}$).
2) The pseudo-parents PP$_i$ of node X_i are X_i's ancestors that are connected to X_i through back-edges (PP$_8 = \{X_1\}$). Notice that PP$_i \neq P_i$.

3) The pseudo-children PC_i of node X_i are X_i's descendants directly connected to X_i through back-edges (e.g., $\mathrm{PC}_0 = \{X_4, X_5, X_1\}$).
4) Sep_i is the separator set of a node X_i: all ancestors of X_i that are connected with X_i or with descendants of X_i (e.g. ,Sep3 $= \{X_1\}$, Sep5 $= \{X_0, X_2\}$). Otherwise stated, given a DFS tree, Sep_i is the minimal set of ancestors of X_i whose removal completely disconnects the subtree rooted at X_i from the rest of the problem. Each node X_i can easily determine its separator Sep_i as the union of separators received from its children, and its parent and pseudo-parents, minus itself.
5) The induced width of a graph G along a given DFS arrangement is equal to the size of the largest separator in the DFS arrangement [43].

Generating DFS trees in a distributed manner is a task that has received a lot of attention, and there are many algorithms available, for example, [43], [44].

B. Principle Operations on DPOP Messages

Two types of messages are exchanged between agents in the DPOP algorithm through the DFS tree.

1) The UTIL message (UTIL_i^j) sent from X_i to X_j is a multidimensional matrix (hypercube) with one dimension for each variable of Separator set of X_i; note that $X_j \in (\mathrm{UTIL}_i^j)$ and $X_i \notin (\mathrm{UTIL}_i^j)$. The UTIL message is a bottom-up process, which starts from the leaves and propagates upwards only through tree edges of the DFS Tree. The agents send UTIL messages to their parents. These messages summarize the influence of the sending agent and its whole subtree on the rest of the problem.
2) The VALUE message is a top-down process, initiated by the root, when UTIL propagation has finished. Each agent determines its optimal value based on the VALUE message it has received from its parent. Then, it dispatches this value to its children through VALUE messages.

Three principle operations used in the DPOP algorithm are the following.

1) Projection($\perp$): $\mathrm{UTIL}_i^j \perp X_k$, the projection of UTIL_i^j hypercube along X_k, is the optimal instantiation of X_k for each instantiation of variables X_i other than X_k ($X_i \in \dim(\mathrm{UTIL}_i^j)$).
2) Join ($\oplus$): Combine two UTIL hypercubes and produce another UTIL hypercube; the aggregation operator in the original DPOP is the sum, but in our case, it can be either the sum, maximum, minimum, product, etc. with respect to composition patterns between the two variables. $\mathrm{UTIL}(X_1, X_2, X_3) \oplus \mathrm{UTIL}\ (X_1, X_2, X_4) \rightarrow \mathrm{UTIL}\ (X_1, X_2, X_3, X_4)$. Each agent X_i has to join hypercubes received from its children ($\mathrm{UTIL}_i^{C_i}$) with hypercubes of its parents and pseudo-parents calculated by X_i ($R_i^{P_i/\mathrm{PP}_i}$) in order to compute the hypercube to send to its parent P_i ($\mathrm{JOIN}_i^{P_i}$).
3) Slice: used in the value propagation where each variable should choose a value that maximize (optimize) its UTIL hypercube and send it to its children.

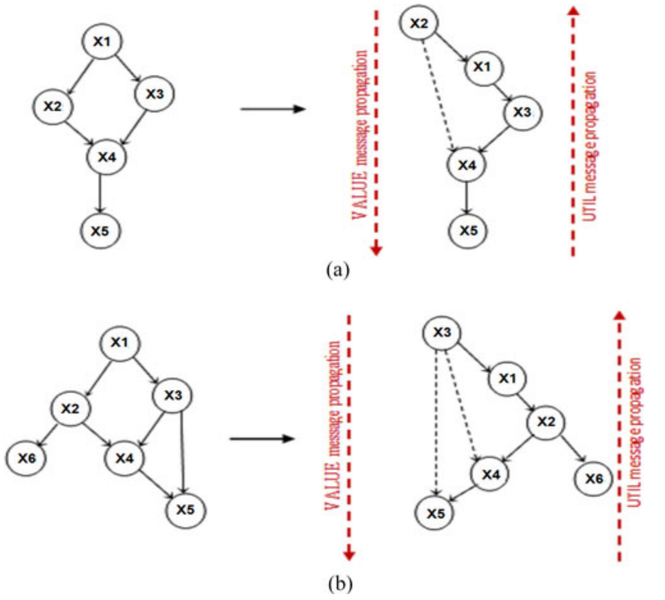

Fig. 3. Examples of composition plans and their corresponding DFS trees required in the DPOP4QoS algorithm.

C. DPOP4QoS Algorithm

We revisit the DPOP algorithm for developing our distributed and optimal Web service selection: the DPOP4QoS. At first, we have to scale QoS values of each service candidate in order to combine and compare correctly the utility values independently from the original scales used by their providers. Therefore, the scaled QoS value $q_{\mathrm{scaled}}(k, j)$ of the kth QoS parameter of a service j is a value between 0 and 1 and is calculated as follows:

$$q_{\mathrm{scaled}}(k,j) = \begin{cases} \dfrac{q_{\max}(k,j) - q_k}{q_{\max}(k,j) - q_{\min}(k,j)}, & \text{if } q_k \text{ is to be maximized} \\ 1 - \dfrac{q_{\max}(k,j) - q_k}{q_{\max}(k,j) - q_{\min}(k,j)}, & \text{otherwise} \end{cases}$$

$q_{\min}\ (k, j)$ and $q_{\max}\ (k, j)$ are, respectively, the minimum and the maximum values of corresponding attributes k in the class of service candidate j and q_k is the original attribute value. We set $q_{\min} = 0$ to avoid division by zero.

In the next step, we have to consider some revisions to the original DPOP algorithm regarding the DFS tree, the join, projection, slice operations, and also how to combine at each agent node hypercubes of children and parent/pseudo-parents (the $R_i^{P_i/\mathrm{PP}_i}$ and $\mathrm{UTIL}_i^{C_i}$ hypercubes) in order to compute the hypercube to send to the parent.

1) In the DFS tree, we have to sort for each node X_i all its neighbors P_i, PP_i, C_i, PC_i, Sep_i coupled with their patterns with X_i according to the global composition plan representing the constraint graph. We give below two examples of composition plan with their corresponding DFS tree (see Fig. 3).
    a) The first one includes one parallel pattern [see Fig. 3(a)]. In the DFS tree of this example, the local view of each node (agent) is as follows:

$X_1 \mapsto PX_1 : (X_2 : \text{Seq}), CX_1 : (X_3 : \text{Seq})$
$X_2 \mapsto CX_2 : (X_1 : \text{Seq}), PCX2 : (X_4 : \text{Seq})$
$X_3 \mapsto PX_3 : (X_1 : \text{Seq}), CX_3 : (X_4 : \text{Seq})$
$X_4 \mapsto PX_4 : (X_3 : \text{Join}), PPX_4 : (X_2 : \text{Join}),$
$CX_4 : (X_5 : \text{Seq})$
$X_5 \mapsto PX_5 : (X_4 : \text{Seq}).$
The induced width of the DFS tree is 2.

b) The second example includes two parallel patterns which increases the induced width of the DFS tree [see Fig. 3(b)]. The DFS local view of each node X_i is as follows:
$X_1 \mapsto PX_1 : (X_3 : \text{Seq}), CX_1 : (X_2 : \text{Seq})$
$X_2 \mapsto CX_2 : (X_4 : \text{Seq}, X_6 : \text{Seq}), PX2 : (X_1 :$ Seq)
$X_3 \mapsto CX_3 : (X_4 : \text{Seq}), PCX3 : (X_4 :$ Seq, X_5 : Seq)
$X_4 \mapsto PX_4 : (X_2 : \text{Join}), PPX_4 : (X_3 :$ Join), $CX_4 : (X_5 : \text{Seq})$
$X_5 \mapsto PX_5 : (X_4 : \text{Join}), PPX_5 : (X_3 : \text{Join})$
$X_6 \mapsto PX_6 : (X_2 : \text{Seq}).$
The induced width of the DFS tree is 3.

2) JOIN operations in the context of Web service composition ($\oplus$): The local utility value UTIL_i^j is calculated according to a) the original DPOP algorithm and b) aggregation formulas for QoS values depending on composition patterns: sequential, parallel, and loop. For this purpose, the JOIN ($\oplus$) operation will be exploded into three sub-JOIN operations: $\oplus^{\text{seq}}$, $\oplus^{\text{Parallel}}$, and $\oplus^{\text{loop}}$.
Definition: If $U = \text{UTIL}_i^j \oplus^{op} \text{UTIL}_k^j$, (op = seq, parallel, or loop), U is also a hypercube with dimension $\dim(U) = \dim(\text{UTIL}_i^j) \cup \dim(\text{UTIL}_k^j)$. For each possible instantiation s of the variables in $\dim(U)$, the corresponding value of $U[s]$ is the sum, maximum, minimum, or the product of the two source hypercubes according to the type of pattern (sequence, parallel, and loop) and the type of QoS (see Table III).
Other composition patterns like the choice (OR Join workflow pattern) can be added to the QoS model by specifying the aggregation function and the JOIN operation would be extended to an additional Join operator.

3) Project operation($\perp$): In the classic DPOP algorithm, the projection of UTIL_i^j matrix along X_k is the optimal instantiation of X_k for each possible instantiation of variables X_i other than X_k. In the case of optimization with satisfaction of constraints, we maintain on each node X_i two hypercubes of utility values: one for saving the global utility values on all QoS attributes UTIL_i^j like in classical version and the second for saving a vector of utility values of each QoS attribute $(\text{UTIL}_i^j)_r$ $(r = 1...L')$ calculated individually. The second hypercube is useful to check QoS constraints; this is done by choosing the optimal values of X_k along X_i axis whose utility values on each attribute $r (r = 1...L')$ satisfy the global attribute constraints Bound_r [according to formulas (4)]; hence, two filtering steps are performed on UTIL_i^j before sending it to the parent.

4) Slice operation: In the slice operation, each variable should also choose a value that maximizes (optimizes) its UTIL hypercube and does not violate any constraints (like in projection step) then send it to its children.

In overall, the DPOP4QoS calculates for each node X_i the hypercube $\text{JOIN}_i^{P_i}$ to send to P_i (X_i parent), as in Algorithm 1:

$$\begin{aligned}\text{JOIN}_i^{P_i} = {} & [(\oplus^{\text{Parallel}} \text{UTIL}_j^{C_i}) \oplus^{\text{Parallel}} (\oplus^{\text{Parallel}} R_i^{\text{PP}_i}) \\ & \oplus^{\text{Parallel}} R_i^{P_i} (\text{ifpattern}(X_j, P_i) = \text{parallel})] \oplus^{\text{seq}} \\ & [(\oplus^{\text{seq}} \text{UTIL}_j^{C_i}) \oplus^{\text{seq}} (\oplus^{\text{seq}} R_i^{\text{PP}_i}) \oplus^{\text{seq}} R_i^{P_i} \\ & (\text{ifpattern}(X_j, P_i) = \text{seq})] \oplus^{\text{seq}} [\text{Loop}_i^N].\end{aligned}$$

1) Cumulate using parallel join ($\oplus^{\text{Parallel}}$) all received UTIL messages from C_i, where pattern (C_i, X_i) = parallel (lines 10, 11, and 15) and then cumulate (with $\oplus^{\text{Parallel}}$) the computed $R_i^{\text{PP}_i}$ and $R_i^{P_i}$, where pattern (X_i, PP_i) = parallel and pattern (X_i, P_i) = parallel (line 18).
2) Cumulate with sequence join ($\oplus^{\text{seq}}$) all received UTIL messages from C_i, where pattern (C_i, X_i) = sequence (lines 10, 11, 12, 13) and then cumulate (with $\oplus^{\text{seq}}$) the computed $R_i^{\text{PP}_i}$ and $R_i^{P_i}$, where pattern (X_i, PP_i) = sequence and pattern (X_i, P_i) = sequence (line 19).
3) Cumulate using sequence join ($\oplus^{\text{seq}}$) all computed parallel join, sequence join, and loop join (lines 6 and 22).
4) Project on X_i according to project operation definition and then send it to P_i (lines 23 and 24).

The DPOP4QoS realizes a complete search by calculating all possible UTIL hypercubes representing all possible combinations of values corresponding to each agent with their neighbors, propagates, and joins all partial solutions through the DFS tree from the leaves to the root. Therefore, the optimality of the algorithm is guaranteed.

The DPOP4QoS algorithm delivers at worst 2*(number of node − 1) messages: one UTIL message by node-1 (the root) and one VALUE message by node-number of leaves. Then, it is linear in the number of variables (nodes), thus producing small communication overheads. However, the size of exchanged messages are hypercubes of dimension = 1 + number of separators of the given sending node, i.e., the size of the hypercube at node $X_i = |X_i| * |X_{i+1}| * \cdots * |X_{\text{number of separators}}|$, where $|X_i|$ is the number of X_i values (the number of concrete services). Thus, the maximum message size is exponential in the induced width of the constraint graph, leading to memory restrictions for problems with large width (nested parallel patterns); in this case, reducing the number of candidate values (concrete services) at each node would alleviate the problem.

VI. Implementation and Experiment Results

The implementation of the service selection system was based on service composition and orchestration middleware the Ubistruct.[2] The platform components are modular, scalable, and

[2][Online]. Available: http://ubistruct.ubiquitous-intelligence.eu/architecture

TABLE III
JOIN OPERATIONS ACCORDING TO THE COMPOSITION PATTERNS S, P, L

	op=Seq ($\oplus^{seq}$)	op=Parallel ($\oplus^{Parallel}$)	op=Loop ($\oplus^{loop}$)
$U[s]$.price	UTIL$_i^j[s]$.Price + UTIL$_k^j[s]$.Price	UTIL$_i^j[s]$.Price + UTIL$_k^j[s]$.Price	$N *$ UTIL$_i^j[s]$.Price
$U[s]$.secur	Min(UTIL$_i^j[s]$.secur, UTIL$_k^j[s]$.secur)	Min(UTIL$_i^j[s]$.secur, UTIL$_k^j[s]$.secur)	UTIL$_i^j[s]$.secur
$U[s]$.EL	UTIL$_i^j[s]$.EL + UTIL$_k^j[s]$.EL/2	Min(UTIL$_i^j[s]$.EL, UTIL$_k^j[s]$.EL	$\sum_{i=0}^{N}$ UTIL$_i^i[s]$.EL
$U[s]$.RE	UTIL$_i^j[s]$.RE * UTIL$_k^j[s]$.RE	(UTIL$_i^j[s]$.RE * UTIL$_k^j[s]$.RE)/2	$\prod_{i=0}^{N}$ UTIL$_i^i[s]$.RE
$U[s]$.RT	UTIL$_i^j[s]$.RT + UTIL$_k^j[s]$.RT	Max(UTIL$_i^j[s]$.RT, UTIL$_k^j[s]$.RT)	$\sum_{i=0}^{N}$ UTIL$_i^i[s]$.RT
$U[s]$.AV	UTIL$_i^j[s]$.AV * UTIL$_k^j[s]$.AV	UTIL$_i^j[s]$.AV * UTIL$_k^j[s]$.AV	$\prod_{i=0}^{N}$ UTIL$_i^i[s]$.AV

Algorithm 1: DPOP4QoS Algorithm.

```
Require: $A_i$ values, $A_i$ Neighbors $P_i$, PP$_i$, $C_i$, PC$_i$
    including their corresponding composition patterns.
 1: procedure UTIL Propagation (for all $X_i$)   $\triangleright$a
    Bottom-up UTIL message propagation
 2:   if $X_i$ is the root then
 3:     skip this
 4:   else
 5:     Calculate (Parallel$R_i$, Seq$R_i$, Loop$R_i(N)$ )
 6:     Seq$R_i^{\mathrm{PP}_i}$=Seq$R_i^{\mathrm{PP}_i}$ $\oplus^{seq}$ $N *$ Loop$R_i$
 7:     ParallelJOIN$_i^{P_i}$ = null
 8:     SeqJOIN$_i^{P_i}$=null
 9:     $**$ calculate UTIL$_i^{P_i}$ the hypercube to send to
        $P_i$ $**$
10:     for all $X_j \in C_i$do   $\triangleright$ if $X_i$ is a leaf skip this
11:       wait for UTIL$_j^i$ message to arrive from $X_j$
12:       if ($X_j$.pattern = seq)then
13:         SeqJOIN$_i^{P_i}$ = SeqJOIN$_i^{P_i}$ $\oplus^{seq}$ UTIL$_j^i$
14:       else
15:         ParallelJOIN$_i^{P_i}$ = ParallelJOIN$_i^{P_i}$
            $\oplus^{Parallel}$UTIL$_j^i$
16:       end if
17:       if ($X_j$ is the last Children)then
18:         ParallelJOIN$_i^{P_i}$ = ParallelJOIN$_i^{P_i}$
            $\oplus^{Parallel}$Parallel$R_i$
19:         SeqJOIN$_i^{P_i}$ = SeqJOIN$_i^{P_i}$ $\oplus^{seq}$ Seq$R_i$
20:       end if
21:     end for
22:     JOIN$_i^{P_i}$ = SeqJOIN$_i^{P_i}$ $\oplus^{seq}$ ParallelJOIN$_i^{P_i}$
23:     UTIL$_i^{P_i}$ = JOIN$_i^{P_i}$ $\perp X_i$   $\triangleright$ optimizing on $X_i$
24:     Send UTIL$_i^{P_i}$ message to $P_i$
25:   end if
26: end procedure
27: procedure Value Propagation (for all $X_i$)   $\triangleright$ a
     Top-down Value message propagation
28:   wait for Value$_i^{P_i}$ message from $P_i$   $\triangleright$ Value
      includes all optimal values of Sep$_i$
29:   $V_i^*$ = slice JOIN$_i^{P_i}$ on Value$_i^{P_i}$   $\triangleright$ find the best $v_i$
      of $X_i$ satisfying constraints according to formula 4
30:   for all $X_j$ $of$ $C_i$do
31:     if $X_i$ is a leafthen
32:       skip
33:     else
34:       Send message Value to (Sep$_i \cup V_i$) to $X_j$
35:     end if
36:   end for
37: end procedure
```

Algorithm 2: Procedure Calculate.

```
38: procedure CALCULATE Parallel$R_i$, Seq$R_i$,
    Loop$R_i(N)$)
39:   $R_i^{P_i}$ is Parent relation of $X_i$
40:   $R_i^{\mathrm{PP}_i}[k]$ are PseudoParent relations of $X_i$,
      $k = 1...nb$ PseudoParents.
41:   Seq$R_i^{\mathrm{PP}_i}$ = $\oplus^{seq} R_i^{\mathrm{PP}_i}[k]$
42:   Parallel$R_i^{\mathrm{PP}_i}$ = $\oplus^{Parallel} R_i^{\mathrm{PP}_i}[k]$
43:   if $P_i$.pattern = seq then
44:     Seq$R_i$ = Seq$R_i^{\mathrm{PP}_i}$ $\oplus^{seq}$ $R_i^{P_i}$
45:   else
46:     Parallel$R_i$ = Parallel$R_i^{\mathrm{PP}_i}$ $\oplus^{seq}$ $R_i^{P_i}$
47:   end if
48:   if $X_i$.pattern = Loop (is a pattern of $X_i$)then
49:     Loop$_i^N$ = $R_i^i$
50:     for $i \leftarrow 1, N-1$ do
51:       Loop$R_i(N)$ = Loop$_i(N)$ $\oplus^{seq}$ $R_i^i$
52:     end for
53:   end if
54: end procedure
```

secure. Moreover, the platform supports multiple communication protocols for IoT such as REST API and XMPP; this enables the integration of a huge number of devices running on different operating systems or in the Cloud. Indeed, REST is a commonly adopted architectural style for the implementation and the communication of the Web services, managing connected objects on the Web. It is based on a simple point-to-point HTTP communication protocol, using format such as JSON or XML. In addition, REST API is widely used by cloud platforms and enables a simple integration with data collection monitoring systems. The XMPP [41] is an open standardized protocol that allows applications to communicate through instantaneous message exchanges on the Web without the need for a middleware or protocol gateways. Another advantage of using XMPP was to integrate the people and devices in the same collaborative ecosystem. The orchestration platform is also based on a distributed database management system over the domains offering the concrete services. For integrating the service selection system into Ubistruct platform, we implemented and assessed through experiments the "DPOP4QoS" Algorithm.

The experiments has been done on a Windows PC, with a Core i3 Processor clocked at 2.1 GHz and 4 GB of RAM. For better system interoperability, our application was implemented

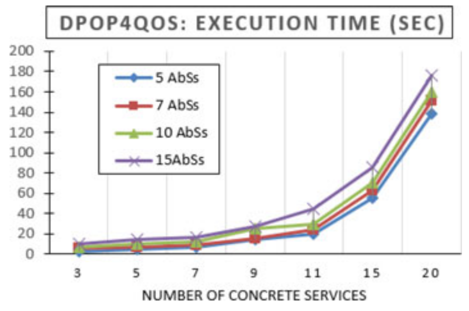

Fig. 4. DPOP4QoS execution time (in seconds).

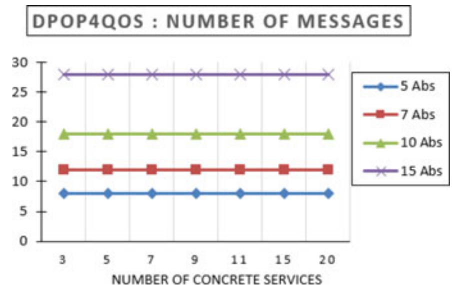

Fig. 5. DPOP4QoS exchanging messages.

in Java under Eclipse. All the selection agents communicate via the XMPP server "The Google Talk." To create an XMPP client for each agent, we used the Smack Java API. Hence, each agent has its own login for the Google Talk Server, its own local orchestration plan, as well as the list of the concrete services. Each concrete service corresponds to specific discrete values of its QoS parameters (a vector of real values). Each agent knows its neighbors in the DFS tree (parent, pseudo-parents, children, pseudo-children) with a specification of the link type with each neighbor (sequential, loop, and parallel). The agents have also access to the user's requirements (global constraints and preferences). We fixed the number of abstract services m to 5, 7, 10, and 15 respectively, and varied the number of concrete services n from 3 to 20. The values of the six QoS parameters have been generated randomly for the n concrete services. The initial scenario of composition used in our experiments is the one explained in the precedent example of Fig. 3(a) with five nodes. For more than five nodes, the structures of the composition are built incrementally by adding one agent each time from the last agents (from the leaves in the DFS tree).

Figs. 4 and 5 depict variations in the number of exchanged messages and execution time of DPOP4QoS (transmission time of a message was between 450 and 460 ms). The obtained measurements show that the algorithm is executed in a very reasonable amount of time (i.e., less than 180 s for 15 abstract services and 20 concrete services per abstract service) and the number of exchanged messages are completely independent of the number of concrete services; one UTIL message by node-1 (the root) and one VALUE message by node-number of leaves (for 15 agents, the number of messages is 15-1(the node) + 15-1(one leave) = 28, as shown in Fig. 5). The variations in the execution time depicted in Fig. 4 is explained by the fact that increasing the number of concrete services involves increasing the size of hypercubes per agent, which slows the processing time (JOIN operations).

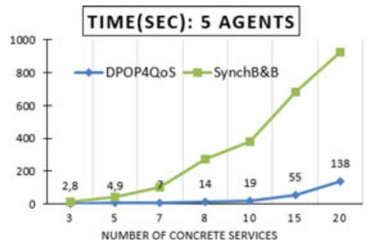

Fig. 6. Execution time (in seconds) in DPOP4QoS versus SynchB&B for five abstract services.

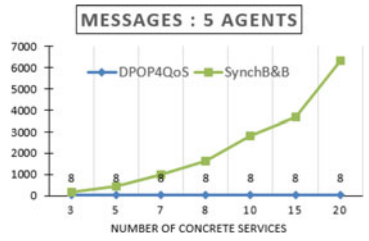

Fig. 7. Exchanged messages in DPOP4QoS versus SynchB&B for five abstract services.

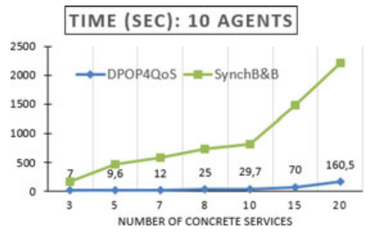

Fig. 8. Execution time (in seconds) in DPOP4QoS versus SynchB&B for ten abstract services.

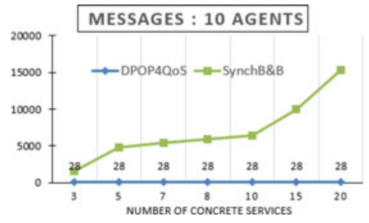

Fig. 9. Exchanged messages in DPOP4QoS versus SynchB&B for ten abstract services.

A. Comparative Results

In this experiment, we aimed to compare the performance of our proposed algorithm to another DCOP algorithm named the synchronous branch & bound (SynchB&B) [31] that we developed for the purpose of these experiments. Like DPOP, the SynchB&B is based on global and complete search. We revisited some aspects of the SynchB&B in order to appropriate it to our problem of Web service selection.

Figs. 6–9 compare the performance of DPOP4QoS and SynchB&B algorithms in terms of the execution time and the number of messages for two problem instances (with five and ten agents) by varying the number of concrete services from 2 to 20. The average transmission delay was 400 ms. It can be observed from Figs. 6–9 and Table IV that the DPOP4QoS far

TABLE IV
DPOP4QoS VERSUS SYNCHB&B: NUMBER OF MESSAGES/EXECUTION TIME

	DPOP4QoS versus SynchB&B: Number of Messages						
Number of CSs	3	5	7	8	10	15	20
DPOP4QoS (five agents)	8	8	8	8	8	8	8
SynchB&B (five agents)	150	430	978	1622	2812	3667	6321
DPOP4QoS (ten agents)	18	18	18	18	18	18	18
SynchB&B (ten agents)	1520	4786	5351	5896	6419	10 023	15 277
DPOP4QoS Improvement	96.7%	98.8%	99.3%	99.6%	99.7%	99.8%	99.8%
	DPOP4QoS versus SynchB&B : Execution time (seconds)						
Number of CSs	3	5	7	8	10	15	20
DPOP4QoS (five agents)	3.1	5.2	7.3	14.1	19	55.2	138.2
SynchB&B (five agents)	15	45	100	273	382	683	923
DPOP4QoS (ten agents)	7	9.6	12	25	29.7	70	160.5
SynchB&B (ten agents)	168	465	587	737	814	1486	2208
DPOP4QoS Improvement	87.5%	93.1%	95.2%	95.7%	95.9%	93.5%	88.8%

outperforms the SynchB&B in terms of the number of messages and the execution time: the reduction in the number of messages reached 99.8% and did not decrease with the number of agents; the reduction in time was between 87.5% and 95.9%, which remains a highly significant result. Hence, unlike SynchB&B, the DPOP4QoS scales perfectly for problems with a large number of abstract/concrete services.

The disadvantage of the DPOP4QoS is that the maximal message size and memory requirements increase exponentially in the induced width of the constraint graph. In other words, if the composite service includes a lot of nested parallel patterns, the induced width in the DFS tree will increase, and hence, the dimension of exchanged hypercubes increases as well. For example, if we had used the scenario of Fig. 3(b), the number of messages would be exactly the same as in the first scenario [see Fig. 3(a)], but the execution time would be slower because the induced width of the generated DFS tree in the scenario [see Fig. 3(b)] is equal to 2, which increases the dimension of hypercubes and hence the processing time of the JOIN operation on each agent node. This problem can be alleviated by removing useless services from each abstract service set using the Pareto search (as the largest hypercube size $= |X_k|^{\text{inducedWidth}}$, where $|X_k|$ is the number of candidates services at node X_k having the largest number of separators).

VII. CONCLUSION

In this paper, we addressed the problem of global and decentralized Web service selection in the context of Web service choreography architecture for IoT environment. The proposed solution is based on the multiagent paradigm and the DCOP formalism. Our objective has been to propose a scalable approach while achieving an optimal solution at the same time. This was very challenging as the majority of the proposed DCOP algorithms use an exponential number of exchanged messages and very high execution time [9]. For that, we revisited a DCOP algorithm to develop an efficient solution, the DPOP4QoS, that uses the dynamic programming optimization technique and generates only a linear number of messages. The algorithm is based on a complete utility propagation method, and hence, optimality is guaranteed. The results of experiments show a very satisfying performance of the DPOP4QoS in terms of timeliness and the number of exchanged messages, which makes it an appropriate solution for real-time distributed service selection problems. However, the size of hypercubes generated at each node must be reduced; this seems very feasible by pruning candidate services using Pareto-dominance techniques or by compacting UTIL messages.

Furthermore, in the context of dynamic service composition and execution, different changes can occur during the execution of a composite service as it may take a long time: QoS values and service availability differ from design-time assumptions. In our ongoing work, we are investigating possible optimization to DPOP4QoS, which makes it more responsive to changes by increasing the reusability of previous computation and by limiting the propagation of new messages upon perturbations. In addition, we plan to extend the solution to manage multiple related service compositions, where each agent has to resolve locally a subproblem of Web service selection while optimizing the global solution. Finally, a comparison with other newer DCOP algorithms is envisaged as future work; for instance, concurrent forward bounding for DCOPs [45].

REFERENCES

[1] A. Jula, E. Sundararajan, and Z. Othman, "Review: Cloud computing service composition: A systematic literature review," *Expert Syst. Appl.*, vol. 41, no. 8, pp. 3809–3824, Jun. 2014.

[2] L. Zeng and B. Benatallah, "QoS-aware middleware for Web services composition," *IEEE Trans. Softw. Eng.*, vol. 30, no. 5, pp. 311–327, May 2004.

[3] T. Yu, Y. Zhang, and K.-J. Lin, "Efficient algorithms for Web services selection with end-to-end QoS constraints," *ACM Trans. Web*, vol. 1, no. 1, 2007, Art. no. 6.

[4] D. Ardagna and B. Pernici, "Adaptive service composition in flexible processes," *IEEE Trans. Softw. Eng.*, vol. 33, no. 6, pp. 369–384, Jun. 2007.

[5] N. M. Lécué, "Towards scalability of quality driven semantic Web service composition," in *Proc. IEEE Int. Conf. Web Services*, 2009, pp. 469–476.

[6] M. Alrifai and T. Risse, "Combining global optimization with local selection for efficient QoS-aware service composition," in *Proc. 18th Int. Conf. World Wide Web*, New York, NY, USA, 2009, pp. 881–890.

[7] J. Li, Y. Zhao, M. Liu, H. Sun, and D. Ma, "An adaptive heuristic approach for distributed QoS-based service composition," in *Proc. IEEE Symp. Comput. Commun.*, Jun. 2010, pp. 687–694.

[8] N. B. Mabrouk, N. Georgantas, and V. Issarny, "Set-based bi-level optimisation for QoS-aware service composition in ubiquitous environments," in *Proc. IEEE Int. Conf. Web Services*, Jun. 2015, pp. 25–32.

[9] A. R. Leite, F. Enembreck, and J.-P. A. Barths, "Distributed constraint optimization problems: Review and perspectives," *Expert Syst. Appl.*, vol. 41, no. 11, pp. 5139–5157, 2014.

[10] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo, "Adopt: Asynchronous distributed constraint optimization with quality guarantees," *Artif. Intell.*, vol. 161, pp. 149–180, 2005.

[11] A. Petcu and B. Faltings, "A scalable method for multiagent constraint optimization," in *Proc. 19th Int. Joint Conf. Artif. Intell.*, Scoltland, U.K., 2005, pp. 266–271.

[12] A. B. Hassine, S. Matsubara, and T. Ishida, "A constraint-based approach to horizontal Web service composition," in *Proc. 5th Int. Conf. Semantic Web*, 2006, pp. 130–143.

[13] W. T. Tsai, P. Zhong, X. Bai, and J. Elston, "Dependence-guided service composition for user-centric SOA," *IEEE Syst. J.*, vol. 8, no. 3, pp. 889–899, Sep. 2014.

[14] Q. Z. Sheng, X. Qiao, A. V. Vasilakos, C. Szaboa, S. Bournea, and X. Xu, "Web services composition: A decades overview," *Inf. Sci.*, vol. 280, pp. 218–238, Oct. 2014.

[15] Y. Wang and J. Vassileva, "Toward trust and reputation based Web service selection: A survey," *Int. Trans. Syst. Sci. Appl.*, vol. 3, no. 9, pp. 118–132, 2007.

[16] Y. Vanrompay, P. Rigole, and Y. Berbers, "Genetic algorithm-based optimization of service composition and deployment," in *Proc. 3rd Int. Workshop Services Integr. Pervasive Environ.*, New York, NY, USA, 2008, pp. 13–18.

[17] X. Wu, X. Xiong, J. Ying, and C. Yu, "QoS-driven global optimization approach for large-scale Web services composition," *J. Comput.*, vol. 6, no. 7, pp. 1452–1460, 2011.

[18] N. Sasikaladevi and L. Arockiam, "Genetic approach for service selection problem in composite Web service," *Int. J. Comput. Appl.*, vol. 44, no. 4, pp. 22–29, 2012.

[19] M. Chen and S. Ludwig, "Fuzzy-guided genetic algorithm applied to the Web service selection problem," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, 2012, pp. 1–8.

[20] W. Yang *et al.*, "A hybrid particle swarm optimization algorithm for service selection problem in the cloud," *Int. J. Grid Distrib. Comput.*, vol. 7, no. 4, pp. 1–10, 2014.

[21] F. Chen, R. Dou, M. Li, and H. Wu, "A flexible QoS-aware Web service composition method by multi-objective optimization in cloud manufacturing," *Comput. Ind. Eng.*, vol. 99, pp. 423–431, 2016.

[22] Z. Z. Liu, D. H. Chu, Z. P. Jia, J. Shen, and L. Wang, "Two-stage approach for reliable dynamic Web service composition," *Knowl.-Based Syst.*, vol. 97, pp. 123–143, 2016.

[23] Y. Chen, J. Huang, C. Lin, and J. Hu, "A partial selection methodology for efficient QoS-aware service composition," *IEEE Trans. Services Comput.*, vol. 8, no. 3, pp. 384–397, May 2015.

[24] M. Zhang, R. Ranjan, M. Menzel, S. Nepal, P. Strazdins, W. Jie, and L. Wang, "An infrastructure service recommendation system for cloud applications with real-time QoS requirement constraints," *IEEE Syst. J.*, to be published.

[25] T. Wu, W. Dou, C. Hu, and J. Chen, "Service mining for trusted service composition in cross-cloud environment," *IEEE Syst. J.*, to be published.

[26] M. Alrifai, T. Risse, and W. Nejdl, "A hybrid approach for efficient Web service composition with end-to-end QoS constraints," *ACM Trans. Web*, vol. 6, no. 7, 2012, Art. no. 7.

[27] Z. Ye, X. Zhou, and A. Bouguettaya, "Genetic algorithm based QoS-aware service compositions in cloud computing," in *Database Systems for Advanced Applications*. Berlin, Germany: Springer, 2011, pp. 321–334.

[28] A. Jula, Z. Othman, and E. Sundararajan, "A hybrid imperialist competitive gravitational attraction search algorithm to optimize cloud service composition," in *Proc. IEEE Workshop Memetic Comput.*, 2013, pp. 37–43.

[29] Z. Zheng, X. Wu, Y. Zhang, M. Lyu, and J. Wang, "QoS ranking prediction for cloud services," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1213–1222, Jun. 2013.

[30] E. Yokoo, "Distributed constraint satisfaction for formalizing distributed problem solving," in *Proc. Int. Conf. Distrib. Comput. Syst.*, Yokohama, Japan, 1992, pp. 614–621.

[31] Y. Hirayama and M. Yokoo, "Distributed partial constraint satisfaction problem," in *Proc. Int. Conf. Principles Practice Constraint Program.*, 1997, pp. 222–236.

[32] R. Mailler and V. Lesser, "Solving distributed constraint optimization problems using cooperative mediation," in *Proc. 3rd Int. Joint Conf. Auton. Agents Multiagent Syst.*, Jul. 2004, pp. 438–445.

[33] W. Zhang, G. Wang, Z. Xing, and L. Wittenburg, "Distributed stochastic search and distributed breakout: Properties, comparison and applications to constraint optimization problems in sensor networks," *Artif. Intell.*, vol. 161, nos. 1/2, pp. 55–87, Jan. 2005.

[34] R. Junges and A. L. C. Bazzan, "Evaluating the performance of DCOP algorithms in a real world, dynamic problem," in *Proc. 7th Int. Joint Conf. Auton. Agents Multiagent Syst.*, 2008, pp. 599–606.

[35] S. Krco, B. Pokric, and F. Carrez, "Designing IoT architecture (s): A European perspective," in *Proc. IEEE World Forum InternetThings*, 2014, pp. 79–84.

[36] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols and applications," *IEEE Commun. Surv. Tuts.*, vol. 17, no. 4, pp. 2347–2376, Fourth Quarter 2015.

[37] R. Khan, S. U. Khan, R. Zaheer, and S. Khan, "Future Internet: The internet of things architecture, possible applications and key challenges," in *Proc. 10th Int. Conf. Frontiers Inf. Technol.*, 2012, pp. 257–260.

[38] Z. Yang, Y. Peng, Y. Yue, X. Wang, Y. Yang, and W. Liu, "Study and application on the architecture and key technologies for IOT," in *Proc. Int. Conf. Multimedia Technol.*, 2011, pp. 747–751.

[39] M. Wu, T.-L. Lu, F.-Y. Ling, L. Sun, and H.-Y. Du, "Research on the architecture of internet of things," in *Proc. 3rd Int. Conf. Adv. Comput. Theory Eng.*, 2010, vol. 5, pp. V5-484–V5-487.

[40] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Univ. California, Irvine, CA, USA, 2000.

[41] P. Saint-Andre, "Extensible messaging and presence protocol (XMPP): Core," IETF, Request for Comment 6120, Mar. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6120.txt

[42] Yachir, "Composition dynamique de services sensibles au contexte dans les systèmes intelligents ambiants," Ph.D. dissertation, Univ. Sci. Technol. Houari Boumediene, Bab Ezzouar, Algeria/Univ. Paris-Est Créteil, Créteil, France, 2013.

[43] A. Petcu, "A class of algorithms for distributed constraint optimization," Ph.D. dissertation, Ecole Polytechnique Federale de Lausanne, Lausanne, Switzerland, 2007.

[44] Z. Collin, "Self-stabilizing depth-first search," *Inf. Process. Lett.*, vol. 49, no. 6, pp. 301–305, 1994.

[45] A. Netzer, A. Grubshtein, and A. Meisels, "Concurrent forward bounding for distributed constraint optimization problems," *Artif. Intell.*, vol. 193, pp. 186–216, 2012.

Nacera Temglit received the Eng. degree and Postgraduate degree "Magister" from Science and Technology University Houari Boumediene (USTHB), Bab Ezzouar, Algeria. She is currently working toward the Ph.D. degree at the Computer Engineering Laboratory, USTHB, and the Laboratory of Images, Signals and Intelligent Systems, University of Paris Est-Créteil, Créteil, France.

She was a Research Engineer with the Research Center on Scientific and Technical Information, Algiers, Algeria. She is an Assistant Professor "Maitre Assistant" with the National High School for Computer Science (ESI), Algiers. Her research interests include modeling, composing, and optimization of Web services in ubiquitous environment.

Abdelghani Chibani received the M.Sc. degree from Paris 6 UMPC University, Paris, France, and the Ph.D. degree from Paris-12 (UPEC) University, Paris, both in computer science.

He is an Associate Professor with University Paris Est-Créteil, Créteil, France, where he is also a Project Manager with the Laboratory of Images, Signals and Intelligent Systems. Since 2005, he has been a Principal Investigator involved with more than 13 EU projects including leadership of large pilots. His research interests include services composition, context awareness, and multiagent systems for ubiquitous computing, Internet of Things, and cloud robotics.

Dr. Chibani was the recipient of the Gold Award in 2011 for the best achievement of the ITEA2 MULTIPOL EU project on multidomain semantic policy management.

Karim Djouani is a Professor, Scientist, and Technical Group Supervisor of soft computing, telecommunication, networking systems, and Robotics. Since January 2011, he has been a Full Professor with University Paris Est-Créteil, Créteil, France, and Tshwane University of Technology (TUT), Pretoria, South Africa. From July 2008 to December 2010, he was seconded by the French Ministry of Higher Education to the French South African Institute of Technology, TUT. He was also a Manager of national and European projects with the Laboratory of Images, Signals and Intelligent Systems, University of Paris Est-Créteil. He is the SARChI Chair in Enabled Environment and Assistive Living at TUT. He has authored or co-authored more than 200 papers in archival journals and conference proceedings. His research interests include the development of novel and highly efficient algorithms for reasoning systems with uncertainty as well as optimization, for distributed systems, networked control systems, wireless ad-hoc network, wireless and mobile communication, and wireless sensors networks as well as robotics.

Mohamed Ahmed Nacer received the Ph.D. degree in computer science from the High School, Polytechnic National Institute, Grenoble, France, in 1994.

He is a Full Professor with Science and Technology University Houari Boumediene, Bab Ezzouar, Algeria, where he is in charge of the software engineering team of the Computer Engineering Laboratory. His current research interests include process modeling, software-architecture-based components, knowledge management, and Web services.