QoS-Aware Automatic Web Service Composition based on cooperative agents

Zaki Brahmi  
RIADI Laboratory, Manouba University  
Higher Institute of Computer Sciences and Communication Techniques of Hammam Sousse  
Sousse University, Tunisia  
e-mail: zakibrahmi@yahoo.fr

M.M. Gammoudi  
RIADI Laboratory, Manouba University  
Higher Institute of Multimedia Arts, Manouba  
Tunisia  
e-mail: mohamed.gamoudi@fst.rnu.tn

Abstract—Automatic Web Service Composition (AWSC) is the processes of combining a chain of connected atomic services together in order to create a more complex and value-added composite service. To differentiate web services which have the same functionalities, Quality of Service (QoS) has been mostly applied. Given a high dynamical and a rapid growth in the number of similar functionally web services, finding an efficient web service composition in a reasonable time satisfying a user requirements has become a challenging task. Many approaches in literature address the problem of QoS-aware automatic Web service composition. However, the majority of the existing approaches are restricted to predefined workflows and have limitations in terms of accuracy, scalability as well as dynamism. In this paper, we propose a novel approach that solves major limitations encountered in the studied approaches. The proposed approach which is a set of cooperative autonomous agents is based in two mains ideas: i) Self-organization of agents into dependency graph named social network agent, and 2) distributed computing of the optimal web services composition by a cooperative protocol among agents. Our approach can generate an accurate composition in a dynamic environment and is scale with the number of web services.

Keywords—web service; automatic composition; agent; cooperative; social network.

I. INTRODUCTION

Web services are software components characterized by their modularity, self-describing, self-containing, and platform-independent. They can be published by service providers through the Internet. Since Web services became available, many organizations prefer to keep only their principal business, but outsource other application services over the Internet [21]. In most cases, an atomic web service does not satisfy the user query [7, 13]. Thus, combining a set of web services is needed in order to create a more complex and value-added composite service, called composite service. Such service (or composition) needs to be: i) Feasible: this by satisfying all user query requirement, ii) Optimal: the service compositions must have good quality of service (QoS) in terms of speed, cost, reliability, and many other measures. The evaluation QoS of a composite service is made by aggregation functions of QoS values of its components. These functions are different from a composition pattern to another and represent the structure of workflow composition [7, 8, 9].

Web service environment is characterized by two mains aspects: large number of web services and higher dynamicity, e.g., during the task execution, events may have occurred, such as invoked service is suspended or its QoS metric is changed. So, service engineers need an efficient automatic service composition, in term of scalability and accuracy.

Several approaches have been proposed in the literature to resolve the problem of QoS-driven automatic web service composition [7, 8, 9, 14, 18]. These approaches are based on genetic algorithm [9, 1], Multiple Criteria Decision Making [7], autonomous agents [19, 20 and 22], heuristic [17, 14], Integer programming [7] etc. However, most proposed approaches consider the search of feasible composition (automatic composition problem) and of the ones that give the optimal QoS (service selection problem) as two separate sub-problems of the general QoS-aware automatic service composition. Additionally, these approaches suffer from scalability and dynamism.

As presented in [13], this paper addresses the QoS-driven automatic service composition problem which merges the automatic service composition problem and service selection problem together. In this work, we propose a novel approach based on a cooperative Multi-agent System (MAS). Our idea is to distribute the computing of the optimal feasible web services composition among a set of cooperatives agents. The proposed approach is based on two main steps: i) Self-organization of agents into dependency graph named social network agent. Contrary to [20], this step is build at the design time which accelerates the computing time of the allocation process. 2) Distributed computing of the optimal web services composition. The first step is done at design-time in order to manage the services dynamicity and to satisfy many user queries simultaneously. The second step is fully distributed among agents with the minimum of communication and a grant of finding the optimal allocation in a polynomial time. Our approach is characterized by accuracy, scalability, and dynamism.

The remainder of this paper is organized as follows, in Section 2; we make a study on the related work as well as their potential limitations. We focus in Section 3 on the basic used concepts and the social network composition web service formulation problem. We describe our proposed approach in Section 4. Section 5 shows the evaluation of our
approach. Finally, we conclude our work and we refer to our prospects.

II. RELATED WORK

QoS-aware automatic web service composition has attracted the attention of many researchers. Principally, the proposed approaches are based on Artificial Intelligence and Operational Research that treat the problem as a search in a graph [8], [12], [13].

In [7], the authors propose a platform named “AgFlow” based on local and global optimization, respectively, based on the approach MCDM (Multiple Criteria Decision Making) and on Integer Programming [7]. Several authors have proposed the use of the genetic algorithm [11], [9], [10], [11]. In [9], genetic algorithm is applied by representing services by genomes. In [14], the authors propose a decentralized approach using a heuristic algorithm for decomposing the problem into sub-problems. These problems will be solved independently. In [8], the authors adopt the heuristic algorithm of ant colony to solve the composition problem. The dynamic criterion ants can treat the web service environment dynamicity problem.

All approaches mentioned above assume that the Work Plan is predefined called abstract composition. For each task in the work plan there is a class of candidate concrete services. The goal is to select the optimal service from each class while maximizing the overall QoS. In [12] and [13] the authors propose an approach that simultaneously addresses the search for correct work plan (automatic service composition) and finding the one that maximizes the QoS (service QoS selection). In [12], the authors treat the problem as a breadth-first search in a dependency graph using an algorithm named “Worklist”. In [13], the authors use the algorithm “Sim-Dijkstra”. In [15], the authors use a search algorithm back to find the correct composition that meets only the functional needs of the user. This approach is based on the grouping of services according to their Inputs and Outputs to reduce the search graph.

The motivation of agent-based service composition [20, 22] is in a sense similar to our approach, which focuses on repetitive coalition formation for service composition according to private preferences. However, these approaches have low scalability. As example in [20] a Distributed Planning Algorithm for Web Service Composition called DPAWSC is proposed. This algorithm is based on the contract-net protocol used by user agent and services agents to build the optimal composition.

Our approach differs from proposed approaches in two aspects: 1) A formal social network of service agent model is proposed, which encapsulate agent technologies and Web service technologies into a cohesive entity. 2) A distributed algorithm for automatic Web service composition is proposed, which based on two layers: design time and runtime. Contrary to [20] the proposed algorithm is fully at runtime.

III. SOCIAL QOS-AWARE AUTOMATIC WEB SERVICE COMPOSITION PROBLEM

In our framework, we consider a user request that need to have a response as a web service composition. The completion of a composition yields a certain value, and it requires varying numbers of web services of different functionalities and QoS. A user request task can be completed only if all required user constraints are fulfilled.

In our approach web service with similar functionalities are grouped in a Service Class (SC). Each SC is managed by an agent nomad Class Agent. In addition, each agent is connected to a limited number of other agents, yielding a social network. For the completion of her task, an agent may enlist the resources of other agents, but only those she’s connected in the network. More formally, let \( W = \{ W_i | 1 \leq i \leq n \} \) denote a set of n Web services.

Definition 1 (Web service): For our approach, a web service \( W_i \) is characterized by its Inputs \( I_{W_i} \), Outputs \( O_{W_i} \), and QoS \( Q_{W_i} \). The Inputs represent the information (preconditions) needed to use the service. The Outputs represent the information (effects) generated from the use of the service. Formally a web service \( W_i \) is defined as (1):

\[
W_i = (I_{W_i}, O_{W_i}, Q_{W_i}).
\]

Definition 2 (Query): A query \( R = (I^R, O^R) \) is a virtual web service. The Inputs \( I^R \) represents information that the user can provided. The Outputs \( O^R \) represents the results required by the user. As example \( I^R \) can be a document “.WORD” and a file “.PDF”. As \( O^R \) we wants to convert the input parameters (“.WORD” and “.PDF”) format images “.GIF”.

Definition 3 (Class Agent): An Class Agent \( CA_i \) is an agent managing a set \( CWS_i \subseteq WS \) of web of services that provide similar functionalities (Input and Output) with possibly different QoS; formally (2):

\[
CA_i = (ID_i, CWS_i, I_{CA_i}, O_{CA_i})
\]

Definition 4 (Social network): An agent social network \( SN = (V, ACE) \) is a dependency graph; where vertices \( V \) are agents class, and each edge \( (i, j) \in ACE \) indicates the existence of a social connection between agents class \( i \) and \( j \).

Definition 5 (Social connection): Two agents \( CA_i \) and \( CA_j \) are connected by a social connection according to the following rule: \( CA_i \) is matched to the agent \( CA_j \) if and only if there exist, some outputs of \( AC_i \) that can match some inputs of \( CA_j \), formally (3):

\[
O_{CA_i} \cap I_{CA_j} \neq \emptyset
\]

Definition 6 (Web Service Composition): A web services Composition \( \beta \) is defined by the tuple \( \beta = (WS_\beta, P_\beta, QoS_\beta) \), where \( WS_\beta = \{ W_i | 1 \leq i \leq m \} \subseteq W \) a sub-set of selected web service to form the composition and \( P \) define an invoking order over \( WS_\beta \). We can model \( P \) as a BPEL
process. The two Web services $W_i$ and $W_m$ represent, respectively, the start and the end node of the composition. The invocation order among these services can be in any patterns form: sequence, split, and join. $\beta$ is feasible if it satisfies the following conditions:

1. $\{ I^* U_{l=1}^{m-1} O_{W_l} \} \supseteq I_{W_{l+1}}$
2. $\{ U_{l=1}^{m} O_{W_l} \} \supseteq O^*$

The $QoS_\beta$ value is the sum of the QoS of each atomic service:

$$QoS_\beta = \sum_{l=1}^{m} W_i$$  \hspace{1cm} (3)

Definition 7 (Efficient web service composition) We say a Web Services Composition $\beta$ is efficient if it is feasible and optimal (the $QoS_\beta$ is the best).

Definition 8 (QoS-Aware Automatic Service Composition problem): Generally, the QoS-aware automatic web service composition problem (QoSAWSCP) can be formally defined as a tuple $QoSAWSCP = <W, R, \phi>$, Where:

- $W$ is a set of web services.
- $R$ represents the user query
- $\phi$ the function (algorithm or approach) that compute the efficient composition $\beta$; $\phi(W, R): W \times R \rightarrow \beta$

IV. COOPERATIVE AGENTS BASED-AUTOMATIC WEB SERVICE COMPOSITION

In this section, we introduce the whole process of solving QoS-aware automatic service composition problem. The core of our approach is a cooperative protocol among agent class CA. The proposed approach named DeCAAWSC (for Decentralized approach for Cooperative Agents based Web Services Composition) is based on a cooperative Multi-agent System (MAS). Our idea is to distribute the computing of the optimal web services composition among a set of cooperatives agents. Our main motivation in using MAS is to take advantage of autonomous agents, and self-organization agents to manage web services and composition dynamicity.

In the next section, the structure and functionality of our approach is presented at first. Next, we describe the application of the proposed approach on a case study.

A. Structural description

Our approach is based on two main ideas: i) self-organization of agents in a social network agent in a design-time. This allows, finding in polynomial time, efficient web service composition and manager web service dynamicity. ii) Satisfying many users' requests simultaneously. The proposed approach is composed of three main layers as shown in the following figure (Fig. 4):

The Service Layer: This layer represents the web service providers and it's the interface between the system and service providers. Second layer is Client Layer; it has an intermediary role between the client and the composition layer. It facilitates the communication between these two parties. Finally, the Layer composition is the main layer in our approach. It represents the service composition engine and contains three types of agents: creator Agent, Class Agent and Composer Agent.

a) Creator Agent: This agent detects the birth of a new class of service. In other words, it allows the organization of services according to their functionality. For each class of services, it creates a Class Agent responsible for the management of this class.

b) Class Agent: Our approach is composed by a set of $n$ CA agents that cooperates together to compute the efficient composition. Each CA has two roles: i) managing a service class (Definition 3) and ii) performing its local composition (Definition 9).

c) Composer Agent: This agent is the main entity in our approach. It has two roles. The first is initializing the composition process to fulfill the client request by triggering the set of agents class to start the computing of composition process. The second role is computing the global optimal composition by selecting the best local composition sent by a set $\phi$ of class agent: $\phi:= \{CA, O_{CA} = O^* \text{ and } 1 \leq i \leq n\}$. In the next section, we describe the functionality of our approach.
B. Our approach functionally

In this section, we present the functionally of our approach based on cooperative agents. The composition process requires two steps: self-organization of class agent and computing the optimal composition by a cooperative protocol among class agents CA.

1) Self-organization of Class Agent:

Agents are organized as a social network agent in the design-time. This in order to fulfill two goals: i) Satisfying many user queries simultaneously. Indeed, social network agent is built independently of user query. ii) Accelerate the time of computing the optimal composition. As defined in Definition 3, class agents are linked based on Input and Output of their services. The self-organization processes beyond the scope of this work.

2) Computing optimal web services composition:

In run-time class agent and composer agent cooperate together to generate the efficient composition. This step is based on computing a set of local composition (Definition 3) by class and the global composition by the composer agent.

Computing the local composition: At design-time, each class agent CAi sorts its set of web service CWS, according to their QoS. At run-time, when it receives a set of local composition βi sent by a set of agent class which has a social connection with ACi, it starts computing its local composition using the algorithm Local_WSComposition.

Algorithm1 Local_WSComposition

Input: - set CAi, CWS, of sorted similar web services.
- S := ∅; set of local composition
- D := ∅; set of similar local composition that their end nod web service which have the same input.
- D := ∅ set of distinct local composition: D = S

Output: Local composition βi

1. bestWS := first(CWS)
2. if |βi| = 1 then
   βi, QoSβi := βi, QoSβi + bestWS.QoS
   βW := βW + |bestWS|
   βi, P := update(βi, P, bestWS)

Else
   (D, S) := Similar(βi)
   BLC := bestLComposition(S)
   D := D ∪ BLC
   βi, QoSβi := ∑βi, QoSβi + bestWS.QoS
   βi, P := update(D, bestWS)
End If

Similar(βi) function builds the two sets S and D. The function update(D, bestWS) merge the set D of local composition and the selected web service bestWS to build the local composition. We can use the language BPEL to build the composition.

Computing the global composition: The global web service composition process is based on the following cooperative protocol:

1. Initially, the user agent sends a query R to the composer agent.
2. The composer builds the set 0 of Class agents that their Input ICA matches the user query Input R : θ = {ICA, IACi ∩ IR = 0}. The composer agent checks whether the set θ is empty. If it is true, then notify the user that its query R cannot be achieved, otherwise it sends the message request(IDca, R) to each agent CA, ∈ θ
3. If class agent CAi receives the message request, then it computes its local composition (or sub-composition) βi = (Wβi, Pβi, QoSβi). In this step, Wβi = {Wj|Wj ∈ CAi, CWS} is the best web service from the set of services CWS in the class managed by CAi. Then it sends the message MyComposition(CAi, βi, R) to all successors agent based on its social network.
4. If the class agent receives a set of MyComposition messages, it computes its local composition by adding the QoS of best service in its class to the best received local composition (if it receives more than one local composition). If the output of its class matched the output of the user query then send its composition to the composer agent. Else it sends its local composition to its successor and Go to 3).
5. Finally, the composer agent computes the global composition and sends it to the user agent. To compute the global composition, composer agent use the algorithm implemented by agent class without line 1.

Example: To better understand the functioning of our system, we apply our solution on social network agents represented as a graph in the following figure (figure 2):

In this graph, nodes represent “Class” agent and each agent is responsible for three services. For example the agent A9 is responsible for 3 services {S90, S91, S92}. S90, S91 and S92 have a QoS respectively equal to 30, 20, and 10. We assume that the client sends the request R = {Inputs=A; Outputs=F}. 

Figure 2. Example of social network agents
In other words, the client provides the parameter A as Inputs and wants to get the parameter F as a result. In the following (Table 2), we present the order of triggering agents and their generated plans.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Sender</th>
<th>Receivers</th>
<th>Message Contain</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Compositor</td>
<td>A0,A4</td>
<td>Null#0</td>
</tr>
<tr>
<td>2</td>
<td>A0</td>
<td>A2</td>
<td>{(S01, A0),#10}</td>
</tr>
<tr>
<td>3</td>
<td>A2</td>
<td>A3,A5,A6</td>
<td>{(S31, A4),#10}</td>
</tr>
<tr>
<td>4</td>
<td>A3</td>
<td>A4</td>
<td>{(S42, A4),#40}</td>
</tr>
<tr>
<td>5</td>
<td>A5</td>
<td>A7,A9</td>
<td>{(S42, A4),#20}</td>
</tr>
<tr>
<td>6</td>
<td>A6</td>
<td>A8</td>
<td>{(S42, A6),#40}</td>
</tr>
</tbody>
</table>

For each iteration, we listed the triggered agents, their message and the receiver of the message. The message must contain the list of candidate’s agents to the composition and the cumulative QoS from the initial agent to the current agent. For example, at iteration 3, the agent A2 sends a message to agents A3, A5 and A6. The message contains ({{S01, A0}, {S23, A2}}#40) indicate that this local message to agents A3, A5 and A6. The message contains QoS equal to 40. This message is the one who has the best QoS are respectively equal to 60, 40 and 20. Agents A8, A7 and A3 send their final composition to the “Compositor” agent. The latter choose the composition that has the best QoS (P3) which generated by A3.

C. Dynamicity Managements

Web service environment is characterized by it higher dynamicity. Therefore, services-based systems are inherently vulnerable to exceptions and the birth of a new web service. In our approach, the management of dynamicity is realized by two types of agents: “Creator” agent and the set of “Class” agents.

The “Creator” agent is responsible for the set QoS information of each atomic service and the discovery of new services. Thus, if the “Creator” agent detects the birth of a new web service $W_p$, it looks for the set of class Agent to which he belongs by checking the following formula (5):

$$\{AC_i | I_{W_p} \cap I_{AC_i} = I_{AC_i} \text{ and } O_{W_p} \cap O_{AC_i} = I_{AC_i}\}. \tag{5}$$

Then it requests appropriates agents class to add it. In cases where any no suitable class to this new service, the “Creator” agent creates a new “Class” agent by assigning to it this service. If there is any change in the values of QoS, the “Creator” agent indicates this to suitable “Class” agent. For example, if a provider service increases the execution price of its service, then the “Creator” agent must report the change to the suitable “Class” agent.

After generating the optimal composition, the user begins to invoke the composite web service by invoking its composing web services. During the invocation process it may have an event occurs. An event can be: i) new Web services that are offered with a better QoS and ii) web services disappear and will not be available. Any event affects the value of overall QoS and the feasibility of the composition. In this case, the “Class” agent must provided the similar service that has the best QoS in the service class. In this step we referred to the work in [16].

V. COMPLEXITY ANALYSIS

To analyse the theoretical time computing of our approach we assume that $p$ (=$|AC,CWS|$, $m$ and $n$ are the number of, respectively, web service per class service, class agent and web services The complexity of our approach depends of:

i. The complexity of the algorithm used to make the sort set of web service,
ii. The local web service composition computing
iii. The agent’s time waiting to receive MyComposition message

For the sort algorithm, the best complexity is evaluated to $O(p\log p)$. This step is done in the design-time.

The complexity of the local composition process as described in the algorithm 1, is based on the Similar function complexity. This function builds form the list $\beta^l$ two set D and S, which represent respectively, the set of similar local composition and the set of distinct local composition. To do this agent class AC, read through $\beta^l$ to find D and S. The size of $\beta^l$ is equal to the number of agent class predecessor of AC. In the the worst case this number is $m$. But in this case, all agents class their input match the input of query user, so no social connection between agent classes. Thus, the complexity of this step is evaluated to $O(m)$.

Each agent class CA, waits, in the worst case, a number of agents which is equal to $m-i$. Only the last agent, that it Input match the output of the user request, waits $m-1$ agents. Therefore, the waiting complexity of an agent $CA_i$ is the waiting sum of agents $CA_{i-1}, CA_{i-2},...,CA_1$; Where, $W_i$ is the waiting time of an agent which equals to the time duration to computing a local web service composition. The $CA_i$’s complexity computing local composition process, in the worst case, is evaluated to $O(\sum_{i=1}^{m} m)=$ $O( m^2(i-1))$. The computing of the final composition is done by the composer agent. In the worst case, $i=m$. This means that each agent class has only one agent predecessor, so the local composition complexity is evaluated to $O(1)$ and the global complexity is evaluated to $O(m)$. In this way, the complexity of our approach is polynomial, in the worst case, it’s evaluated to $O(m^2)$. In reality, in the context of our work, this complexity will be never happened.
VI. IMPLEMENTATION AND EVALUATION

The experiments are performed on an Intel® Core™ i3 CPU @ 2.53 GHz with 4 GB of RAM. To implement the various agents of our system, we use the Multi-Agents platform JADE installed on ECLIPSE. Firstly, we create $n$ Class agents denoted $CA_1, CA_2, \ldots, CA_n$, and $m$ web services shared on the set Class agent: $|CA_i, CWS| = m/n$. Our experiments show the evolution of the execution time of our system. To simplify the understanding of the functioning of our system, we used the response time of services as a single QoS metric. The response time values are generated randomly. The results of these experiments are presented in the Fig. 3 and Fig. 4.

![Figure 3. Execution time of our approach with different services number per 20 class.](image-url)

In Fig. 3 (and Fig. 4) we show two curves representing the evolution of the execution time depending on the number of services per class. The first curve shows the evolution of the execution time of our approach and the second represents the evolution of the execution time of the approach using Dijkstra algorithm [13]. The scalability of our system allows it to generate an optimal exact composition in terms of QoS in a time that does not exceed 200 (ms). This composition is generated from 10,000 services available. We use 20 service classes and for each class we use 1000 candidate services. In Fig. 5 we show the evolution of execution time of our approach and the approach using Sim-Dijkstra algorithm [13] based on the number of class while fixing the number of candidate services to 1000 services per class. For these experiments, we vary the number of class between 5 and 20 classes.

![Figure 5. Execution time of our approach with different class number (time displayed on logarithmic scale in base 10).](image-url)

VII. CONCLUSION AND FUTURE WORKS

In our present work, we propose an efficient web service composition approach based on Multi-Agents System. These agents cooperate together to find the optimal and feasible composition. Our solution is based on two steps: 1) Self-organization of agents as a social network. Each class agent managed a set of service sharing the same functional proprieties. This in order to minimize the time computing and 2) distribute the web services composition computing among class agents. This approach is characterized by accuracy and reacts to dynamicity of web service. Our approach is scalable while keeping the constraint of minimization the QoS. In our future works, we will plan to use Case-based reasoning (CBR) concept to improve our approach.

REFERENCES


