Towards a Multi-tenancy aware Cloud Service Composition

Wael Sellami
ReDCAD Laboratory
University of Sfax, Tunisia
wael.sellami@gmail.com

Hatem Hadj Kacem
ReDCAD Laboratory
University of Sfax, Tunisia
hatem.hadjkacem@fsegs.rnu.tn

Ahmed Hadj Kacem
ReDCAD Laboratory
University of Sfax, Tunisia
ahmed.Hadjkacem@fsegs.rnu.tn

Abstract—Cloud computing has gained a momentum by combining multi-tenant and scalable services in order to obtain a composed service and some added value. Therefore, it is important for composed cloud services to support multi-tenancy during the services matching. For this purpose, we tackle the problem of integrating multi-tenancy in cloud services composition and we determine crucial issues that can be solved. To do this, we propose a multi-tenant aware services composition approach to facilitate the dynamic services matching at run-time. This research is based on service patterns which are used within the composition process for integrating multi-tenancy architecture and reducing the difficulty of cloud services matching. Furthermore, we propose a middleware layer (Composition as a Service) for implementing our approach into conventional cloud architecture. The performed evaluation proves the efficiency of our approach for services based applications in the cloud computing.

Keywords – Cloud service composition, PaaS, Multi-tenancy awareness, Service Patterns.

I. INTRODUCTION

Nowadays, cloud computing has emerged as a new concept to adopt new economic models and to offer services over scale environment. According to NIST [3], cloud computing is a model that aims to supply, deliver and consume software and physical resources on demand through Internet protocols. Indeed, the cloud is a stack of services commonly referring to Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). Therefore, the cloud computing poses new challenges such as the development of complex applications based on services matching to satisfy the increasing demands of consumers. More concretely, cloud services composition consists in assembling a set of elementary software and hardware services to create a new composite service. Thus, composed services may be characterized by a set of properties such as scalability/elasticity, data persistence, multi-tenancy and so on. Indeed, the multi-tenancy [2] is a key concept of cloud computing where a single instance of a service serves multiple users/organisations known also as tenants. It allows cloud services composition according two modes [6]: native multi-tenancy and multiple instances. These two modes correspond to the multi-tenant aware application and they may affect all technological stack (application, middleware, virtual machine and infrastructure) of cloud services composition. Indeed, multi-tenant awareness means that the same instance of the composed service must be flexible to operate with different multi-tenancy modes relied on the requirements and contexts data of tenants. The contextual information can be the location of the tenant or any other one related to his environment.

When addressed to service based applications in the cloud, composition faces crucial challenges. First, it is imperative to consider the multi-tenancy architecture to attribute the business process for managing the scalable services instances. Second, composition solutions have to be awareness through using the multiple instances mode for the involved services while retaining the native multi-tenancy one for the others.

Many attempts to provide dynamic cloud services composition in the cloud exist but most of them [4] [12] [13] have focused on a cloud services matching aspect and ignored the multi-tenancy key concept in composed service. Moreover, at the best of our knowledge, all of these matching solutions either do not take care at all of scalability issue, or do not resolve the flexibility when the number of tenants increases.

In this paper, we propose a new approach to provide on-demand the multi-tenancy aware services composition in the cloud computing. In order to meet the services matching, multi-tenant service patterns are proposed as abstract model which provide reusable and suitable instances for cloud services. In addition, an algorithm for matching multi-tenant cloud services is defined through the use of the proposed service patterns. To enable delivery of on-demand multi-tenant service composition, we encapsulated our approach in a middleware MTA4SC (A Multi-Tenancy Aware Middleware For Cloud Services Composition) which is located between the SaaS and the PaaS levels of cloud architecture.

The rest of this paper is organized as follows: In Section II, we introduce a background for cloud services composition. In Section III, we present the problem of integrating the multi-tenancy within the composed services. The proposed multi-tenant aware composition approach is presented in Section IV. Thereafter, Section V presents a middleware to illustrate our approach followed by the experiment results. Related work are presented in Section VI. Finally, Section VII concludes this paper and highlights some future work.
II. BACKGROUND

A. Services Composition in Cloud Computing

The cloud services composition consist in assembling different software services and utility computing services into a coherent single service to provide a higher value service. Indeed, the composition may be achieved in vertical and horizontal dimensions [8]. Vertical services composition denotes the coordination of multiple architectural layers including enterprise service buses, application servers, operating systems and so on. Horizontal services composition deals with the combination of the uniform services layer such as APIs, libraries and hosting services of PaaS layer. However, composition services in cloud computing is different from service oriented architecture and utility computing in terms of web or grid service form. Indeed, services in cloud are encapsulated like an abstract entity that is delivered in different levels of services (combination of software, platforms or hardware). Thus, composition in the softwares layer is similar to the web service matching. Whereas, the composition in the utility layer is similar to the services combination in grid computing. Additionally, it is hard to use traditional service matching methods by the characteristics of dynamically configured, massively scalable, distributed resources and multi-tenancy architecture of cloud services. Due to the differences between these architectures, service composition in cloud computing environments must support: (i) The dynamic matching of massively scalable services, (ii) The flexibility of the composed service due to changing of users requirements, and (iii) The main properties of cloud computing like on-demand services, driven by economies of scale and multi-tenancy.

B. Multi-tenancy: definition and support

The multi-tenancy [2] is considered as an essential property of cloud services. Indeed, it is an architecture in which a single instance of a service serves multiple users/organisations. Each one is identified as a tenant. This latter can be a single user or a whole company consisting of several distinct users. The multi-tenancy attempts to replace many small services instances with one or few great instances in order to maximize the utilization of infrastructures. In particular, a tenant should trust cloud services providers based on isolation between private data and virtual machines regardless of his concerns or goals. Therefore, the multi-tenancy is based on two basic modes [6] which are:

- Native multi-tenancy: it means that all tenants are supported by a single shared service instance over various hosting resources. Indeed, the same instance denotes that the same service using the same code on the same infrastructure is used by different tenants.
- Multiple instances: it denotes that each tenant has his dedicated service instance over a shared hardware, operating system or a middleware server in a hosting environment. A service instance should require a tenant specific behaviour for a service.

In this context, the multi-tenancy awareness aims to deal with different multi-tenancy modes in order to provide a transparent interaction between each tenant and the business processes. Otherwise, the tenants do not feel of using a dedicated application and doubt of the security of their data and consequently they may move to other service provider.

III. PROBLEM STATEMENT AND MOTIVATION

The multi-tenancy can affect the composed cloud service in the whole layers during interaction with the atomic services. Indeed, the composed service can be faced with conflicts of selecting the suitable multi-tenancy mode: a native multi-tenancy or multiple instances. Yet, the multiple instances mode can be contrasted with the native multi-tenancy mode. The process can interact with services using an incorrect mode, either the tenant wants a dedicated service instance or the services provider prefers sharing a service for multiple tenants. Moreover, the single instance of the composed service can have the same behavior between all tenants regardless of specific contexts and requirements of each one.

To understand the problem well, we can take a case study originated from our previous work [10] which is a composed web service assisting the customers to sale their equities on an on-line stock market. This example is adapted to the characteristics of this present research in terms of using cloud services and integrating the multi-tenancy property.

Therefore, the execution of this composed service is not only related to functionalities but also related to the different tenant’s multi-dimensional constraints for the interacted services. Let us consider a example, where tenant A requires fast response time despite the high price for such performance services. While tenant B can find response time is less important and is primarily concerned with the security and privacy of the same services functionalities. The composed service provider, on the other hand, has to apply his optimisation goal for sharing service instances in order to save the cost as much as possible. The identified problem in this context is caused by the instantiation mode of interacted services where actors have different goals and only one instance of the composed service is shared on the server.

This challenge leads us to two interesting issues:
1) How can a multi-behavior instance of a dynamic composed service for all tenants be achieved?
2) In which multi-tenancy mode can services hosted at cloud providers be assembled into a composed service?

Therefore, to overcome these issues, our work takes special care in developing a multi-tenancy aware approach as a key building block of the PaaS offer.
IV. Multi-Tenancy Cloud Services Composition Approach

To achieve the multi-tenancy of composed service, it is necessary to resolve tenants issues for the services matching at the run-time. Indeed, the structured requirements of the tenant allows to describe the problems and to reduce the complexity of the composed service. In order to facilitate the business process awareness, a multi-tenant service pattern is adopted. For that, we propose an multi-tenant aware approach for cloud services composition as presented in Figure 1. Before identifying the set of services and integrating the multi-tenancy in the process, a composition schema is required. Hence, we assume that the schema already exists, and we focus on the matching of concrete cloud services that perform the business process. So, the objective of this approach is to create multi-tenancy aware processes by the dynamic matching of service through existing service patterns.

A. Tenant requirements

A tenant for a composed service can be a single user or a whole company consisting of distinct users. The information defined in the requirement specifies the tenant’s goal. Indeed, eliciting the concerns can be viewed as constraints for achieving this goal which is relied on two aspects: functional and non-functional concerns. These concerns are represented by the pairs \( <\text{concept}, \text{value}> \) in which \( \text{concept} \) denotes the required concern and \( \text{value} \) is the weight of this concern. Hence, we formalize a requirement as: \( R = <\text{TID}, \text{G}, \text{C}> \) where:

- \( \text{TID} \): tenant authentication identity.
- \( \text{G} \): the goal of the tenant.
- \( \text{C} \): the concern to express the related properties and it can be represented by the couple \( <\text{FC}, \text{NFC}> \) where:
  - \( \text{FC} = \{f_1, ..., f_n\} \) is the set of functional concerns.
  - \( \text{NFC} = \{nfc_1, ..., nfc_m\} \) is the set of non-functional concerns.

B. Multi-tenant service patterns

We define a multi-tenant service pattern as a generic service which provides a reusable and an abstract description for multi-tenancy modes. It is defined over services and present the ways of matching cloud services to satisfy tenant’s goals. Indeed, patterns [7] can be considered as an intermediate level of abstraction which intend to bridge the gap between the tenant requirements and dynamic composed services. Their definition and management reside essentially at design phase. A service pattern instance is generated at run-time by inputs representing tenant’s goal and concerns. Therefore, multiple instances of these patterns can be created at different granularity going from the cloud services concerns to the to the execution related requirements.

The multi-tenant service patterns are based on a domain ontology which provides definitions for the main concepts and their relationships. The Figure 2 shows the proposed model of multi-tenant service pattern. By using a predefined domain ontology, each service pattern consists of a goal, one or more concerns and some rules. A multi-tenant service pattern is associated with a service mode that can be a native multi-tenancy mode over a multiple instances mode.

In the following, we describe the key concepts that constitute the multi-tenant service pattern model.

- **Goal.** This concept emphasizes the problem that the service pattern solves. It corresponds to the service interface which will be used during services interaction to establish the correspondence between the business process and cloud services allowing to carry it out.
- **Concerns.** It is a broad concept that encompasses a number of functional and non-functional properties such as security, availability and reliability. This concept is an important criterion for the service multi-tenancy mode among a set of services implementing similar functional capabilities.
- **Rules.** They are required to guide the selection of the multi-tenancy services modes for the different services invocations. These rules can be atomic or composite. Flexibility originates from the fact that the composed service is governed by the rule mechanism which is a relevant feature for service providers.
In fact, the variability of suitable multi-tenancy mode provided by the rules will help the provider to increase the awareness degree of the composed service.

C. The matching algorithm

To compose cloud services, a matching algorithm is proposed (see Algorithm 1). This algorithm deals with the assembling of concrete services based on multi-tenant service patterns. Its input parameters are \( iq \) and \( sep \) where \( iq \) is a tenant query and \( sep \) is a cloud service Endpoint. The output is the OID of the involved service instance. Indeed, a set of multi-tenant service patterns is discovered and refined as a starting point for invoking services. It is based on semantic matching using an ontology to establish the correspondence between the goal of the process invoke activities and the goal of the service patterns (cf. lines from 4 to 9 of Algorithm 1). Then, the selected pattern is responsible for driving the invocation of the cloud service relied on the appropriate multi-tenancy mode (cf. lines from 10 to 12 of Algorithm 1). This step uses the concerns provided by the tenant specification to parse the defined rule in the pattern. The fact of the assignment of many services patterns to the business process has some desirable benefits. Specifically, the existence of multiple services patterns introduces a degree of variability and adaptability. Likewise, the flexibility is supported from the fact that the service matching is guided by the rules mechanism which is a key feature to process evolution.

Algorithm 1 TenantServicesMatching(iq, sep)

1: //Variables
2: \( mtmode: \) the multi-tenancy mode;
3: \( serpat: \) pattern;
4: Extract the goal of invoke characterization \( gc \) from \( iq \);
5: Request all patterns \( p \) from service patterns base;
6: For all \( pi \) do
7: Perform a semantic similarity between \((gc, pi)\);
8: Endfor
9: \( serpat= \) select the best similar multi-tenant pattern;
10: Parse the \( serpat \) and analyze the rule;
11: \( mtmode = \) Determine the appropriate multi-tenancy mode;
12: invokeService (\( mtmode, sep \));
13: return the OID of the service instance;

V. EXPERIMENTS

A. Multi-Tenancy Aware Middleware For Cloud Services Composition (MTA4SC)

A middleware is regarded as a software that mediates between two separate services in order to reduce complexity of distributed applications. Thus, we encapsulate the main functionalities of our approach into a MTA4SC middleware which is located between the SaaS and the PaaS levels of cloud architecture as demonstrated in Figure 3. The MTA4SC middleware is a part of our framework presented in [11] for establishing the context-aware cloud service matching and the adaptation of composed cloud services. Indeed, before selecting the set of concrete cloud services and assigning them to tasks involved in the composition process, the specification of this process is first required. To do this, we consider two specifications of service composition: Business Process Execution Language (BPEL) and Service Component Architecture (SCA). In this work, we deal with the case of BPEL. SCA treatment can be deduced later from our proposal. In the following, we present the main components of the middleware and their related communications.

1) Tenants Service: The tenants service layer represents the access point for tenants of a composed service. It deals with the recognition and the extraction of the requirement expressed by the tenant. This module can be deployed on portable multimedia player and mobile device such as Android, SmartPhone and IPhones devices.

2) Peer to Peer discoverer (P2PD): The P2PD adopts the approach proposed by [9] to perform the services discovery in distributed P2P network. Indeed, this step is achieved by matching the semantic similarity between the tenant concerns and services description in the registries. Thus, the P2PD selects the suitable service and identifies its Endpoint.

3) Service Patterns Discoverer (SPD): This module deals with the discovery of multi-tenant service patterns in the patterns database. Indeed, the SPD resolve the discovery query by matching the semantic similarity between the BPEL invoke activity goal and the service pattern goal.
4) Multi-tenant Balancer engine (MTBE): This component is a multi-tenant services executing which is based on Algorithm 1 to bind the services instances to the dynamic composition processes. The MTBE uses the selected patterns received from SPD module. It parses the defined rules to determines the appropriate multi-tenancy mode and tries to invoke the cloud service via its Endpoint. Moreover, the MTBE may create a new identical instance, instead of continuing with the same instance in order to handle the additional load and better satisfy the tenant’s requirements.

B. Evaluation

To validate our work, we carry on exploiting OWL-S description language to implement multi-tenant service patterns in a computer understandable way. For the cloud manager, we used OpenNebula 4.0 [1] which is installed in a private cloud with 6 performed machines (4 VMs in Front-end and 15 VMs in the others nodes). Moreover, for the composition schema, we based on BPEL 2.0 as an abstract business process. For the experiments, we defined two scenarios that reflect the objectives that we want to highlight in our experiments based memory consumption and the response time criteria. So, we evaluated the proposed on-line stock market process before and after adding the MTA4SC middleware to estimate the overhead of the multi-tenancy aware services composition.

In the first series, we generated a number of requests (from 500 to 25000) for 2000 tenants invoking the BPEL process (without middleware) and we took memory consumption measurements for each number. Then, we generated the same number of tenants requests to the same BPEL process enhanced with the middleware module. Figure 4 presents the different values stored during these series. We note that for the obtained curve, the overhead of memory consumption increased regarding the number of requests. Effectively, when the number of requests is between 500 and 25000, we notice a huge number of requests affecting the performance of the composed service without taking advantage of the multi-tenancy architecture. In contrast, integrating the middleware in services matching is virtually reduced with an average of 150 M.O of memory consumption. This reduction is explained by the maximum efficiency of sharing services instances among different tenants.

In the second series of tests, we aimed to measure the latency time of the business process before and after integrating the MTA4SC middleware for the same number of requests of the previous experiment. After storing these measurements, we calculate the average latency time for each series. The stored values are shown in Figure 5. This experiment shows that the latency time of the composed service is linear and it increases exponentially with the number of the sent requests. This is due to the overhead caused by the interaction time between the various services matching that are distributed in the VMs. Furthermore, the variance between the two applications is fair even with the increasing number of requests (an average of 1 second between 20000 and 25000) which explains that the MTA4SC middleware does not influence the matching process and the tenant remains using the application as the only user.

VI. RELATED WORKS

Recent research works have been devoted to matching cloud services and developing approaches and tools, which enable a developer to manage them. In [13], Xiangbing et al. proposed a semantic service composition approach based on cloud computing. They used a Bayesian decision and a graph structure to describe and discover services. Afterwards, they performed a semantic matching algorithm to obtain a chain of services process. Nevertheless, this approach does not provide support for extending the pool of usable services instances. Tsai et al. [12] suggested an approach for matching and testing cloud services. They are based on service interfaces and injected service techniques to cluster services. Thus, they proposed a pull tests such as service-level MapReduce for services checking and ensuring the reliability of matching. However, the complexity of the composed service is increased by the introduction of an injection service technique for every involved target service in the matching. The work presented in [14] addressed the problem of the composed service provi-
sion. For this issue, they transformed the users requirements to data flow graphs and proposed an extensible QoS model to calculate the QoS values of services. In addition, they adopted a genetic algorithm to find the services composition based on chromosomes crossover and mutation. Although, this work does not support the scalability of a single business process for matching multi-tenant services. In their work [4], Garcia et al. discussed the dynamic contracting of service providers and a model based agent for the composition problem in cloud computing. Each participant in the cloud is represented by an agent: consumer, provider, broker, resource. They proposed a dynamic service composition algorithm to facilitate the decentralized services matching. However, the multi-behaviour of the composed service is omitted where the requirements are different for multiple tenants. In [15], Zeng et al. are relied on QoS to discover and filter services, which are described with semantic input and output parameters. Thus, they suggested a Service Matching Algorithm based on the semantic similarity between input and output parameters to achieve this composition. In contrast to our approach, this work lacks the flexibility where the composed service is governed by the rule mechanism which is a relevant for service providers.

Therefore, we notice that the whole related work outlined above has three major drawbacks. First, they do not deal with the integration of the multi-tenancy into composition which is a major concept of the cloud computing. Second, they lack dynamic scalability for managing involved services instances. Third, the multi-behaviour aspect of the same process instance during services matching is not a central concern for these approaches.

VII. CONCLUSIONS AND ONGOING WORK

Due to the open and flexible nature of cloud services, there is a gap between service matching efficiency and the multi-tenancy awareness needs. In this paper, we have proposed an approach that integrates the multi-tenancy architecture within cloud services composition. Indeed, the key contributions include: (i) an algorithm for multi-tenancy aware cloud services matching, (ii) a multi-tenant service pattern as an intermediate model for the services invocation, and (iii) a MTA4SC middleware layer to provide an on-demand service “Composition as a Service” in cloud computing architecture. Finally, we described its implementation and we performed different experimentations with similar non multi-tenant process under equivalent load.

In the future work, we plan to include new features into the MTA4SC middleware to support the deployment of new specifications of service composition such as the Service Component Architecture (SCA). We also aim to extend services patterns to apply adaptations or reconfigurations on composed service at runtime.

REFERENCES