ComputerandInformation Science; Vol. 12, No.1; 2019
ISSN 1913-8989 E-ISSN 1913-8997
Published by Canadian Center of Science and Education

Theoretical and Pragmatic Cases of a Rich-Variant Approach for a User-Aware Multi-Tenant SaaS Applications

Houda Kriouile[1] & Bouchra El Asri[1]

[1] IMS Team, ADMIR Lab, ENSIAS, Rabat IT Center, Mohammed V University in Rabat, Morocco

Correspondence: HoudaKriouile, IMS Team, ADMIR Lab, ENSIAS, Rabat IT Center, Mohammed V University in Rabat, Morocco. Tel: 212-618-156-098. E-mail: houda.kriouile@um5s.net.ma

Received: October 22, 2018 Accepted: November 8, 2018 Online Published: January 3, 2019

doi:10.5539/cis.v12n1p1 URL: https://doi.org/10.5539/cis.v12n1p1

Abstract

Software as a Service cloud computing model favorites the Multi-Tenancy as a key factor to exploit economies of scale. However Multi-Tenancy present several disadvantages. Therein, our approach comes to optimize instances assigned to multi-tenants with a solution using rich-variant components while ensuring more economies of scale and avoiding tenants hesitation to share resources. The paper present the theoretical and pragmatic cases of a user-aware multi-tenancy SaaS approach focused on graph-based algorithms. The theoretical case consists in having a set of tenants while the pragmatic case consists in adding a new tenants to a set of tenants.

Keywords: cloud computing, saas, multi-tenancy, rich-variant component

1. Introduction

Cloud Computing has emerged these last decade as a trend model of computing. It is one of the hottest paradigms of how to build and deliver IT services. Software as a Service (SaaS) is a form of Cloud computing that refers to software distribution model in which applications are hosted by a service provider and made availability to customers over a network. As a key enabler to exploit economies of scale, SaaS promotes Multi-Tenancy (MT), the notion of sharing resources among a large group of customer organizations, called tenants. MT brings several advantages to SaaS, however, it only satisfies requirements that are common to all tenants as well as the fact that tenants themselves hesitate about sharing.

To tackle these problems, a plethora of research work has been performed to facilitate SaaS applications customization according to the tenant-specific requirements. Most of these works are based on exploiting benefits of MT, variability management, and tenants' isolation on a single instance (Ruehl, 2014), (Walraven et al., 2014), (Samrajesh et al., 2016), (Orchei et al., 2016). Likewise, our approach aims to create a flexible and reusable environment enabling greater flexibility and dynamicity for customers while leveraging the economies of scale. The approach is a user-aware solution integrating a functional variability at application components level and deployment variability at multi-tenants end-users level as well. Moreover, the approach focuses on satisfying stakeholders, providers and customers, while maintaining a level of performance and remaining efficient.

The aim of our work is to provide an economy of scale for SaaS application providers while minimizing the cost to its applications tenants. We seek to achieve our goals using multi-variant components that give more possibilities of sharing allowing more instances sharing and over lower cost and better communication between tenants' communities. Through our work we try to response to the following research questions:

Q1: How can customers' deployment requirements be captured?

Q2: How can deployment information be formally represented?

Q3: How can an optimal distribution be deduced?

This paper presents theoretical and pragmatic cases of our work trying to solve the research questions above. It consists in a rich-variant approach for multi-tenant SaaS applications. The remainder of this paper is structured as follows. Section II identifies the problem and determines the context of our work which is instances optimization using components. Section III presents the main contribution of our approach consisting in a graph-based algorithm computing optimal deployment, this section treats the theoretical case consisting in having a set of tenants. Section IV shows the pragmatic case of our approach consisting in adding a new tenant to a set of tenants. Section V

presents and compares several approaches studied as related works. Finally, Section VI gives concluding remarks and directions of future work.

2. Instances Optimization with Component-based Cloud Computing provisioning

The emergence of cloud computing has required more and more variability in term of types of services, types of deployment, and cloud participants different roles. Thus, variability modeling is needed to manage the inherent complexity of cloud systems.

SaaS is a delivery model whose basic idea is to provide on-demand client applications on the Internet. SaaS applications are consumed by many customers who have different requirements. Thus, customers who consume the same application generally have different requirements. This type of requirement usually requires alternative software architectures. In other words, when the requirements of the applications are changed, the software architectures of these applications must be adapted to meet them. As a result, the requirements and architectures have intrinsic variability characteristics.

In addition, other problems are raised by Multi-Tenancy which is favored by SaaS to exploit economies of scale. This means that a single instance of an application serves multiple clients. Customers or tenants are, for example, businesses, clubs or private individuals who have adhered to the use of the application. Even if several clients use the same instance, each one of them feels that the instance is only designated for them. This is archived by isolating tenant data from each other. Unlike single tenancy, Multi-tenancy hosts a plurality of tenants on the same instance.

However, one of the main disadvantages of multi-tenancy applications is the need to ensure the accuracy of all possible configurations of the application in addition to the hesitation of customers to share the infrastructure, the code of the application or data with other tenants. This is because customers are afraid that other tenants may access their data due to a system error, malfunction, or destructive action.

On the other hand, in multi-tenant SaaS applications consumer does not have to worry about doing updates and upgrades, adding security and system patches and ensuring the availability and performance of the service. In addition to this, fast elasticity and pooling of resources are key features of the Cloud (Mell and Grance, 2011), which promote variability in the Cloud Computing environment and in particular for multi-tenant contexts.

Operational cost of the application must decrease by sharing computing resources among the plurality of tenants. It is sought to realize a multi-rental application optimized for the operator and the tenant at the same time. For the operator, the cost and effort must be reduced, especially with respect to the use of the IT resource infrastructure. And for the tenant, the data security needs to be improved at the same time. Once a deployment configuration has been created, there will be one or more instances of each deployment level. The deployment configuration is optimal if it generates a minimal cost, using only a minimum number of units of application component instances and the underlying infrastructure layers.

Cloud operators need less infrastructure to offer an application in the MT model than the Single-Tenancy model. However, the resources required are not the only way to save costs, the operator can also minimize the effort required to maintain a high number of instances (Bezemer and Zaidman, 2010).

In (Fehling and Mietzner, 2011) authors show how applications that were built from components and which use different cloud service models as an execution engine can be composed to form new applications that can be re-offered as service. These applications were designed in the spirit of customization, so their variability was modeled using the application model and variability model of the Framework Cafe (composite application framework). Using these models the Framework derives a personalization flow that guides the application client through configuring this variability using a self-service portal to reflect its individual needs. In addition, an extension of the application model was introduced to model triggers that perform certain actions in the case of certain conditions in the environment or their initiation by the user or at a certain time. Since these triggers have also been targeted by application customization, the individual behavior management can be performed by the Framework to support automatic provisioning and de-provisioning of the application as well as common management functionalities for the applications.

3. Our RV-Cloud Approach: Theoretical Case

3.1 User-Aware Tenancy Approach based on Rich-Variant Component

In order to provide a more flexible, more dynamic and more reusable environment for SaaS application providers, our approach proposesauser-aware tenancy approach based on Rich-Variant Component (RVC).

An RVC is defined as an application building block that encapsulates an atomic functionality. All functionalities and properties that the RVC provides to and requires from other RVCs must be captured by a described interface,

through which all interactions flow. In addition, an RVC has several deployment variants, it can be used in its different ways and therefore changes behavior dynamically depending on the functionality and the end user. Moreover, it is very important for this work, that the RVCs can be deployed independently of each other.

Through our work, we seek to exploit economies of scale while avoiding the problem of customer hesitation to share with others as well as allowing better communication between customer communities.

Our approach proposes a provider platform from which information is exchanged between the provider and his customers. The provider presents his offers and the customers express their needs and requirements.

In addition to collecting customers functional requirements, the main idea of our work is to collect even the deployment sharing requirements. This allows to consider deployment requirements of all tenants when calculating an optimal distribution of application instances over customers renting this application.

The Rich-Variant Architecture of our approach was presented in previous works (Kriouile and El Asri, 2018), it is based on an Execution Framework described in Fig. 1. Our Execution Framework proceed on three steps by applying three algorithms.

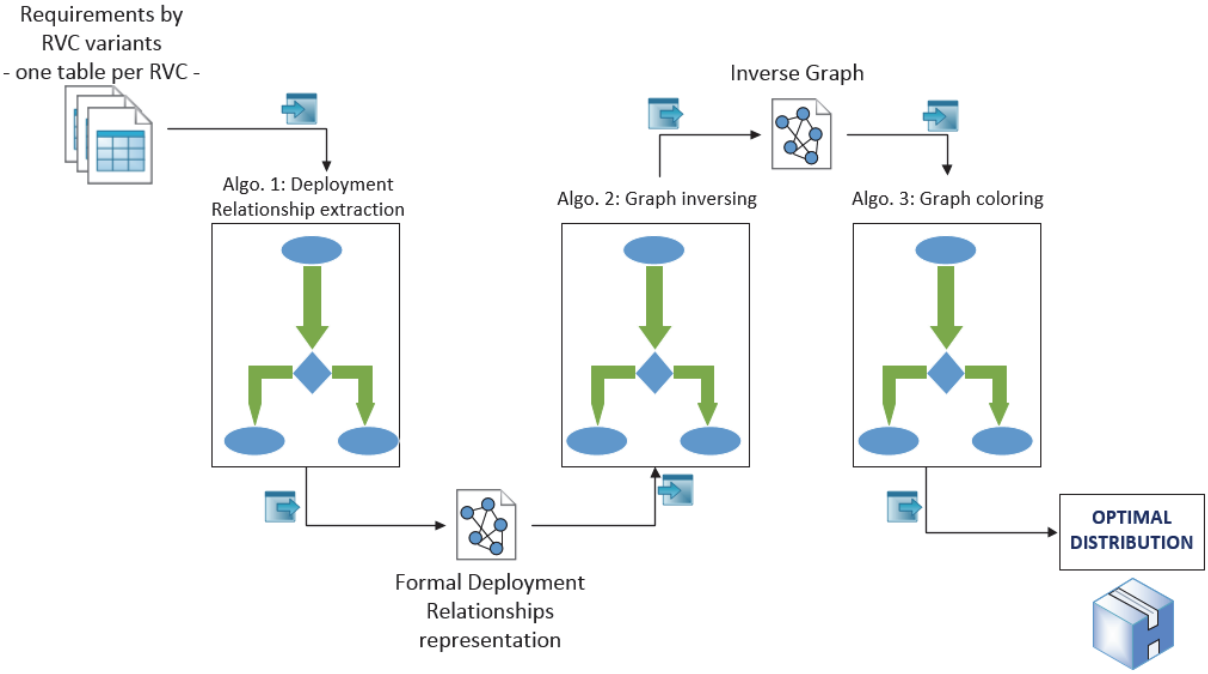

Figure 1. Description of Execution Framework (Kriouile and El Asri, 2018)

From our platform, tenants choose the functionalities they desire to have in the application and specify their deployment requirements for each functionality. When a tenant doesn't precise any deployment requirement for a functionality, it means that he has no problem sharing this functionality. In this case, we consider the default value which is "Share with anyone". In the following, we show how we formalized the expression of deployment requirements to facilitate their capture.

Based on deployment requirements, we deduce deployment relationships that describe which tenants can share which variants of the RVC. We formally represent deployment relationships with Undirected Edge Labeled Graphs, one graph by an RVC. While vertices represent tenants, edges represent if two tenants can share variants or not. Labels on edges indicate the variants involved in sharing relationship represented by the edge. If an edge has no label, it means that sharing relationship concerns the RVC with all its variants. Fig. 2 presents an example of deployment relationships represented by a graph.

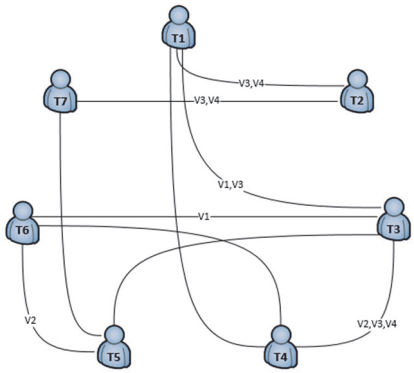

Figure 2. Example of deployment relationshipsgraph

The three steps of our treatment are as follows:

Step 1: Inverse the undirected edge labeled graph

- Keep the same vertices;
- Make each two non-adjacent vertices become adjacent with an unlabeled edge;
- Make each two adjacent and unlabeled vertices become non-adjacent;
- Make each two adjacent and labeled vertices become adjacent with a label containing the complement of variants in the initial label.

For example, for a RVC with five variants V1, V2, V3, V4, and V5, if the original label contains "V2, V5" then the label on the inverse graph is "V1, V3, V4".

Step 2: Divide vertices by RVC variants number

The second step is to divide the vertices by the number of RVC variants. If the number of variants is n, there will be n parts on each vertex each referring to a RVC variants.

Step 3: Color the Inverse Graph

The third step is to color the inverse graph. Our coloring function assigns a color to each part of each vertex so that two adjacent vertices in a variant have different colors each in the part referring to this variant.

Algorithm A formalizes the processing of this step where we start by assigning a color to all parts of a first vertex. Then, for each next vertex, for each part referring to a variant, for each color if the vertex is not adjacent to the vertices of this color according to the variant, then we attribute to it the same color; if the vertex is adjacent to at least one vertex of the color, we move on to the next color. Once you have arrived at the last existing color, if we did not give color to the part of the current summit, we then give it a new color.

This coloring part returns a set of used colors $\mathcal{C}$={C1, ..., Cd}. Each used color is a set of sections of vertices colored by this color.

```
Algorithm A: The Coloring Algorithm
-----------------------------------------------------------------------
Input : m number of tenants T1, ...,Tm,
        and n number of RVC variants V1, ..., Vn
Output : C ={C1, ..., Cd}, with d number of colors used
-----------------------------------------------------------------------
1:  Give a color to T1 (for all variants) and  put d=1
2:  For i from i=2 to i=m do
3:      For j from j=1 to j=n do
4:          For k from k=1 to k=d do
5:              if Ti is not adjacent to any T from Ck according to Vj
6:              then give the color Ck to Ti.Vj and put j=j+1
7:              else    if  k=d
8:                      then put d=d+1
9:                      and give the new color Cd to Ti.Vj
10:                     and put j=j+1
11:                     else put k= k+1
12:                     end if
13:             end if
14:         end For
15:     end For
16: end For
17: return C ={C1, ..., Cd}
```

Considering that when instantiating a RVC according to a variant, we can use the same instance according to the

other variants, we deduce that the number of instances required to complete the deployment is the number of used colors, what means that it is the cardinality of the set C. Moreover, we can also deduce the optimal distribution of these instances on the different tenants, and that from the same return of the coloring function. Indeed, each color C_k designates a specific instance of the RVC and the elements of this color C_k refers to tenants who will use this instance and according to which variant they will use it.

To conclude, our processing, which aims at calculating the valid and optimal deployment for an RVC, can be simplified and concluded in Algorithm B. This algorithm takes as input an undirected graph with labeled edges representing the RVC deployment information, and returns output all the colors used.

```
Algorithm B: Overall Algorithm
-----------------------------------------
Input : G an Undirected Edge Labeled Graph,
        and n the number of variants
Output :C ={C1, ..., Cd}
-----------------------------------------
1: Inverse the graph G to G'
2: Divide the vertices of G' by n part
3: Color the graph G'
4: return C={C1, ..., Cd}
-----------------------------------------
```

3.2 Matrix Modeling

In this section, we present our work in a more formal way using formulas, algorithms and mathematical concepts.

3.2.1 Deployment requirements Capture

In the aim of facilitating the capture of deployment requirements expressed by tenants, we defined four possible cases. Tenants can express their requirements for each application functionality using the following expressions:

- SWAny: Share with anyone (default value)
- SWJ(X): Share with just X ;
- DSW(X): Don't share withX ;
- DSWAny: Don't share with anyone.

Where X can take the values: "P" (as Partners), "C" (as Competitors), "T_i" (for a specific Tenant), or a list of the previous values.

Requirements are ordered in a table where we store the requirements of each tenant for each application functionality. We have a such table for each application. As a result of the translation of requirements concerning functionalities to requirements concerning variants, we obtain a table by RVC containing each tenant requirements for each RVC variant. However, there may be several expressions in one table cell, to settle this problem we apply the following transition rules:

- SWAny and Z give Z
- DSWAny and Z give DSWAny
- DSW(X) and DSW(Y) give DSW(X,Y)
- SWJ(X) and SWJ(Y) give DSWAny
- DSW(X) and SWJ(Y) give SWJ(Y)
- DSW(X) and SWJ(X) give DSWAny

Where Z can take any of the four possible expressions (i.e. Whatever Z).

3.2.2 From requirements to the graph

From this step the work is the same for each RVC, so for the remainder of the paper we keep working on a single RVC. Then, let's havea RVC with n variants. And let m be the number of tenants. We formalize the table of m tenantsrequirements about the n RVC variants by E a two dimensions (m x n) table in which each element e_{ik} is the requirement of tenant i about variant k, as shown by (1):

$$E = (e_{ik}),\ (i=1,...,m,\ k=1,...,n) \quad (1)$$

The deployment relationships Graph is formalized by a Boolean three-dimensional matrix G (m x m x n) where

the g_{ijk} value indicates if tenant i and tenant j may share the variant k, as shown by (2):

$$G = (g_{ijk}),\ (i,\ j=\ 1,...,m,\ k=1,...,n) \tag{2}$$

If the g_{ijk} value is 1 then both tenants i and j can share variant k, and if the g_{ijk} value is 0 then they cannot share. By default, all tenants can share all variants unless they declare the opposite. Therefore, we initiate the g_{ijk} values of the matrix G by 1. Thereafter, we traverse cells of requirements table R and decide whether to change the g_{ijk} value according to the expression of eik.

- If e_{ik} = DSWAny then $g_{ijk} = g_{jik} = 0$ where i and j are different.
- If e_{ik} = SWJ(tenants' LIST) then $g_{ijk} = g_{jik} = 0$ where tenant j does not belong to the LIST and where i and j are different.
- If e_{ik} = DSW(tenants' LIST) then $g_{ijk} = g_{jik} = 0$ where tenant j belongs to the LIST.
- If e_{ik}= SWAny then we change nothing.

This step is formalized by Algorithm 1 thereafter. The end of this step makes the transition from tenant requirements to deployment relationships graph.

```
Algorithm 1 : Extracting Deployment Information
-------------------------------------------------------
INPUT : Table R(m,n)
OUTPUT : Matrix G(m,m,n)
-------------------------------------------------------
    /*              Initialisation              */
    FOR (k=0; k<n; k++)
        FOR (i=0; i<m; i++)
            FOR (j=0; j<m; j++)
                LET gijk=1
    /*              Initialisation              */
    FOR (k=0; k<n; k++)
        FOR (i=0; i<m; i++)
            IF (rik = "DSWAny") THEN
                FOR (j=0; j<m; j++)
                    IF (j!=i) THEN
                        LET gijk=0 and gjik=0
                    ENDIF
            ELSE IF (rik = "SWJ(LIST)") THEN
                FOR (j=0; j<m; j++) DO
                    IF (j NOT IN LIST) & (j!=i) THEN
                        LET gijk=0 and gjik=0
                    ENDIF
            ELSE IF (rik = "DSW(LIST)") THEN
                FOR (j=0; j<m; j++)
                    IF (j IN LIST ) THEN
                        LET gijk=0 and gjik=0
                    ENDIF
            ENDIF
```

3.2.3 From the graph G to its inverse

Thereafter, we pass from the graph G to the inverse graph formalized by a Boolean three-dimensional matrix G(m x m x n) where the g'_{ijk} value takes the opposite of g_{ijk}, as shown in (3):

$$G' = (g'_{ijk}),\ (i,\ j=\ 1,...,m,\ k=1,...,n)\ such\ as\ g'_{ijk} = \begin{cases} 1 & if\ g_{ijk} = 0 \\ 0 & if\ g_{ijk} = 1 \end{cases} \tag{3}$$

Algorithm 2 formalize the transition from graph G to its inverse thegraph G':

```
Algorithm 2 : Graph Inversing
-----------------------------------------------------
INPUT : Matrix G(m,m,n)
OUTPUT : Matrix G'(m,m,n)
-----------------------------------------------------
    FOR (k=0; k<n; k++)
        FOR (i=0; i<m; i++)
            FOR (j=0; j<m; j++)
                IF (gijk=0) THEN
                    LET g'ijk=1
                ELSE LET g'ijk=0
                ENDIF
```

3.2.4 Towards the Optimal Distribution

The optimal distribution of RVC instances is formalized by a two-dimensional matrix D (m x n) where the d_{ik} value takes an integer indicating the color assigned to the part referring to the variant k from the graph vertex referring to the tenant i, as shown by (4):

$$D = (d_{ik}),\ (i=1,...,m,\ k=1,...,n) \tag{4}$$

As we had already explained in the previous chapter, to color the inverse graph we first give a first color to all parts of a first vertex. So as an initialization, we give the value 1 to all elements of the first line of the matrix D, as shown in (5):

$$d_{1k} = 1\ ,\ (k=1,...,n\ ) \tag{5}$$

Let h be the number of used colors, we initiate h at the value 1. And let w and u be indicators initiated to 0. Coloring of the inverse graph is completely formalized by the Algorithm 3 which takes as input the graph G' and gives as output the matrix D. The number of instances required to complete the deployment is the number of used colors, it means that it is the number h. Moreover, we can also derive the optimal distribution of these instances on the various tenants, and that from the matrix D returned by the algorithm. Indeed, each color refers to a specific instance of the RVC and the elements of the matrix D with the same value - referring to the color- show tenants who will use this instance and according to which variant they will use it.

```
Algorithm 3 : Graph Coloring
-----------------------------------------------
INPUT : Matrix G'(m,m,n)
OUTPUT : Matrix D(m,n)
-----------------------------------------------
    /*          Initialisation          */
    FOR (i=0; i<m; i++)
        FOR (k=0; k<n; k++)
            LET dik =1
    LET h <- 1 & w <- 0 & u <- 0
    /*          Initialisation          */
    FOR (i=1; i<m; i++)
        FOR (k=0; k<n; k++)
            LET f<-1;
            WHILE (f<h+1)
                LET j<-0;
                WHILE (j<i)
                    IF (djk = f) THEN
                        IF (g'ijk = 1) THEN
                            LET w<-1 & j<-i
                        ELSE LET j<-j+1
                        ENDIF
                    ELSE LET j<-j+1
                    ENDIF
                IF (w=0) THEN
                    LET dij<-f & u<-1 & f<-h+1
                ELSE LET f<-f+1 & w<-0
                ENDIF
            IF (u = 0) THEN
                LET h<-h+1 & dik<-h
            ELSE LET u<-0
            ENDIF
```

4. Pragmatic case: Adding New Tenant

4.1 Global Algorithm for the pragmatic case

The previous section proposes an algorithm dealing with the situation where the provider wants to calculate the optimal distribution of instances on a set of tenants considered to have requested the rental of the application at the same time. However, the situation that arises in reality is the case where a set of tenants already uses the application according to an optimal distribution calculate previously, and a new tenant requests the lease of the same application. The new problem to be solved is which application instance to give the new tenant taking into account his requirements and his relations with the old tenants. We will call this last case in the rest of the document by the practical case.

Consider then the situation where we have m tenants T_1 to T_m renting an application and a new tenant T_{m+1} wishing to rent the same application. Similar to treatment proposed in the previous section, we work on RVCs building the application one by one. For the pragmatic case, the processing steps on a RVC are as follow:

• Extract the new G_{m+1} graph from the deployment relationships by considering the requirements of the new tenant as well as those of the previous tenants.

• Invert the graph G_{m+1} in G'_{m+1}

• Divide the vertex T_{m+1} of the graph G'_{m+1} into n parts, the vertex corresponding to the tenant T_{m+1}

• Color the vertex T_{m+1} according to Algorithm A for i = m+1

The formalization of these steps is presented by Algorithm C below.

```
Algorithm C: Overall Algorithm for case of adding new tenant Tm+1
-----------------------------------------------------------------
Input: Gm graph of deployment relationships of m tenants,
       et Cm={C1, ..., Cd} set of colors used for the m tenants,
       et Requirements of the new tenant Tm+1
Output: Cm+1={C1, ..., Cd}
-----------------------------------------------------------------
1: Found the new deployment relationships graph Gm+1
2: Inverse Gm+1 to  G'm+1
2: Divised G'm+1 vertices by n parts
3: Color the edge Tm+1 of graph G'm+1
4: return Cm+1={C1, ..., Cd}
-----------------------------------------------------------------
```

4.2 Matrix Modeling for the Pragmatic Case

In this section, we are still working on a RVC component with n variants. In the same way that we did the matrix modeling in the previous sections, we formalize in this section the table of new tenant T_{m+1}requirements, we also formalize the graph ofdeployment relationship between the m+1 tenants, as well as its inverse graph, and finally we formalize the optimal distribution of RVC instances on m+1 tenants.

We formalize the table of the requirements of the new T_{m+1} tenant concerning the n variants of the RVC by an array E_{m+1} with an dimension with n elements in which each element ek is the requirement of the tenant T_{m+1} concerning the variant k, as shown by (6).

$$E_{m+1} = (e_k), \ (k=1,...,n) \tag{6}$$

The graph of the deployment relations between the m+1 tenants is formalized by a three-dimensional Boolean matrix G_{m+1} (m+1 x m+1 xn) where the g_{ijk} value indicates whether the tenant i and the tenant j can share the variant k, as shown by (7).

$$G_{m+1} = (g_{ijk}), (i, j= 1,...,m+1, k=1,...,n) \tag{7}$$

If the value g_{ijk} is equal to 1, then both tenants i and j can share the variant k; and if the value g_{ijk} is equal to 0, then they cannot share it. By default, all tenants can share all variants, if they say otherwise. Thus, we introduce the g_{im+1k} and g_{m+1jk} new values of the matrix G_{m+1} in 1. Then, we go through the boxes of the table of requirements E of dimension (m+1 x n), requirements of the m+1 tenants, and we decide if we should change the value g_{ijk} according to the expression of e_{ik}.

• If e_{ik} = DSWA then $g_{ijk} = g_{jik} = 0$ for cases where i and j are different.

• If e_{ik} = SWJ (LIST OF renters) then $g_{ijk} = g_{jik} = 0$ for cases where tenant j does not belong to the LIST and where i and j are different.

• If e_{ik} = DSW (LIST OF TENANTS) then $g_{ijk} = g_{jik} = 0$ for the case where the tenant j belongs to the LIST.

• If e_{ik} = SWA then we do not change anything.

The next step goes from the graph G_{m+1} to the inverse graph also formalized by a Boolean matrix G'_{m+1} with three dimensions (m+1 x m+1 xn) where the value g'_{ijk} takes the opposite of value g_{ijk} as shown (8).

$$G'_{m+1} = (g'_{ijk}),\ (i,\ j = 1,...,m+1,\ k=1,...,n)\ such\ as\ g'_{ijk} = \begin{cases} 1 & if\ g_{ijk} = 0 \\ 0 & if\ g_{ijk} = 1 \end{cases} \quad (8)$$

The new optimal distribution of RVC instances is formalized in the same way - with one more rendering - by a two-dimensional D_{m+1} matrix (m+1 xn) in which the d_{ik} value takes an integer indicating the color assigned to the part referring to the variant k of the top of the graph referring to the tenant i. Formalization shown by (9).

$$D = (d_{ik}),\ (i=1,...,m+1,\ k=1,...,n) \quad (9)$$

In this pragmatic case, the d_{ik} values for i of 1 to m are known and taken as input of Algorithm 4. It remains to find the d_{ik} values for i = m + 1.Let h be the number of colors used initiated at 1. And let w and u be indicator indices initiated at 0.Algorithm 4 below formalizes all the treatment of calculating the optimal distribution to give to the new tenant.

```
Algorithm 4 : optimal distribution for case ofadding new tenant
-----------------------------------------------------------------
Input: Matrix G(m,m,n),
       et Matrix D(m,n),
       et Table Em+1(1,n)
Output: Matrix D(m+1,n)
-----------------------------------------------------------------
    For (k=0; k<n; k++)
        For (i=0; i<m+1; i++)
            do gimk = 1
        For (j=0; j<m+1; j++)
            do gmjk = 1
    do h <- 1 & w <- 0 & u <- 0
    For (k=0; k<n; k++)
        For (i=0; i<m+1; i++)
            if (rik = "DSWA") then
                For (j=0; j<m; j++)
                    if (j!=i) THEN  do gijk=0 et gjik=0
                    EndIf
            Else if (rik = "SWJ(LIST)") THEN
                For (j=0; j<m+1; j++) do
                    if (j don't exist in LIST) & (j!=i) THEN
                        do gijk=0 and gjik=0
                    EndIf
            Else if (rik = "DSW(LIST)") THEN do
                For (j=0; j<m+1; j++)
                    if (j from LIST) THEN do gijk=0 et gjik=0
                    EndIf
            EndIf
    /*          Inversing Gm+1 to G'm+1          */
    For (k=0; k<n; k++)
        For (i=0; i<m+1; i++)
            For (j=0; j<m+1; j++)
                if (gijk=0) THEN do g'ijk=1
                Else do g'ijk=0
                EndIf
    /*              Computing Dm+1               */
    do i=m
        For (k=0; k<n; k++)
            do  f<-1;
            while (f<h+1)
                do j<-0;
                while (j<i)
                    if (djk = f) do
                        if (g'ijk = 1) THEN do w<-1 & j<-i
                        Else do j<-j+1
                        EndIf
                    Else do j<-j+1
                    EndIf
                if (w=0) THEN do dij<-f & u<-1 & f<-h+1
                Else do f<-f+1 & w<-0
                EndIf
            if (u = 0) THEN do h<-h+1 & dik<-h
            Else do u<-0
            EndIf
```

5. Related Works

Several research works have been performed in the context of architectural patterns for developing and deploying customizable multi-tenant applications for Cloud environment. Several approaches from those - cited below - were studied and compared in Table 1. The comparison is based on common characteristics shared by the studied approaches.

- Approach A: (Composite as a Service (CaaS) (Fehling and Mietzner, 2011)) shows how applications built of components, using different Cloud service models, can be composed to form new applications that can be offered as a new service.
- Approach B: (Matchmaking of IaaS Offers Leveraging Linked Data (Zaremba et al., 2013)) presents models of Expressive Search Requests and Service Offer Descriptions allowing matchmaking of highly configurable services that are dynamic and depend on request.
- Approach C: (Service line engineering (Walraven et al., 2014)) presents an integrated service engineering method, that supports co-existing tenant-specific configurations and that facilitates the development and management of customizable, multi-tenant SaaS applications.
- Approach D: (Mixed-tenancy Systems (Ruehl, 2014)) addresses the deployment variability based on the SaaS tenants requirements about sharing infrastructure, application codes or data with other tenants. It proposes a hybrid solution between multi-tenancy and simple tenancy.

The new notion brought by our approach and that is not proposed by the others approaches is the roles accessibility based on the concept of Multiview. All cited approaches aim to improve flexibility and reusability in their ways. In the aim of exploiting economies of scale, some approaches rely on the multi-tenancy, we do the same in our approach but, in addition, we benefit from the use of Multiview notion to exploit more and more economies of scale.

Table 1. A Comparative Study on Customizable Approaches For Cloud Environment

Approach	Approach A	Approach B	Approach C	Approach D	Our Approach
Cloud application area	SaaS	IaaS, Service Computing	SaaS	SaaS	SaaS
Variability	-Functional -Deployment	Deployment	Functional	-Deployment -Functional	-Functional -Deployment
Accessibility by roles	Not proposed	Not proposed	Not proposed	Not proposed	Use of Multiview concept
Flexibility	Dynamically scale based on customer demand	Service consumer might specify a flexible search request using enumerations and ranges	Use of Service line and Workflows	Flexibility to use depending to the tenant using the application	Flexibility according to tenants, and flexibility according to enabled view
Reusability	Use of component-based software	Service Variant Hierarchy promotes reuse	Modular middleware layer	Use of application component	Use of RVCs
Economies of scale	Use of highly flexible templates enabling	Not proposed	Application level multi-tenancy	Mixed tenancy (hybrid solution between multi-tenancy and	Multi-tenancy and Multiview notion

increasing customers base	simple tenancy)

6. Conclusion

Flexibility, dynamicity, and reusability are challenging issues for multi-tenancy SaaS applications. In this regard, our user-aware SaaS approach consists in integrating two types of variability to create a more flexible and reusable SaaS environment while exploiting economies of scale and avoiding the problem of tenants hesitation about sharing with others. In this context, this paper addresses the algorithmic part formalization, which aims to compute a valid and optimal RVC instances distribution on tenants while respecting their deployment requirements. For this purpose, we first presented the context and motivations of the problem. Then, we presented our User-Aware SaaS Approach. Then, we treated the formalization of our approach using some mathematics concepts. Finally, to illustrate our model we applied our algorithm to a case study. As future work, we think about projecting our approach in the domain of Model-driven engineering for a more modern and more general vision.

References

Bezemer, C. P., &Zaidman, A. (2010). Multi-tenant SaaS applications: maintenance dream or nightmare? In IWPSE-EVOL'10, Antwerp, Belgium, 20-21 September 2010, pp. 88-92. https://doi.org/10.1145/1862372.1862393

Fehling, C., &Mietzner, R. (2011). Composite as a Service: Cloud Application Structures, Provisioning, and Management, *It - Information Technology Special Issue: Cloud Computing*, April, 188-194. https://doi.org/10.1524/itit.2011.0642

Kriouile, H., & El Asri, B. (2018). A Rich-Variant Architecture for a User-Aware multi-tenant SaaS approach. *International Journal of Computer Science Issues, 15*(4), 46-53. https://doi.org/10.5281/zenodo.1346047

Mell, P., &Grance, T. (2011). The NIST Definition of Cloud Computing. National Institute of Standards and Technology", In: Information Technology Laboratory, Version 15 (September 2011), 10–7. https://dx.doi.org/10.6028/NIST.SP.800-145

Orchei, L.C., Petrovski, A., & Bass, J. M. (2016). Optimizing the Deployment of Cloud-hosted Application Components for Guaranteeing Multitenancy Isolation. in International Conference on Information Society (i-Society), p. 77. https://doi.org/10.1109/i-Society.2016.7854180

Ruehl, S. T. (2014). Mixed-Tenancy Systems A hybrid Approach between Single and Multi-Tenancy, doctoral diss., Department of Informatics, Clausthal University of Technology, June. http://www.dr.hut-verlag.de/978-3-8439-1664-6.html

Samrajesh, M. D., Gopalan, N. P., & Suresh, S. (2016). A scalable component model for multi-tenant SaaS application.In *Int. J. Advanced Intelligence Paradigms*,8(2).https://doi.org/10.1504/IJAIP.2016.075727

Walraven, S., Landuyt, D. V., Truyen, E., Handekyn, K., &Joosen, W. (2014). Efficient customization of multi-tenant Software-as-a-Service applications with service lines. *Journal of Systems and Software*,*91,* 48-62. https://doi.org/10.1016/j.jss.2014.01.021

Zaremba, M., Bhiri, S., Vitvar, T., & Hauswirth, M. (2013). Matchmaking of IaaS cloud computing offers leveraging linked data, *Proc. the 28th Annual ACM Symposium on Applied Computing (SAC)*, New York, USA, 383-388. https://doi.org/10.1145/2480362.2480440

Copyrights