Table des matières

Liste des figures

Liste des tableaux

Introduction générale

Le calcul distribuée à base de services et les paradigmes de développement associées, y compris l'architecture orientée services (SOA), les services Web, ou l'idée de «*Software as a Service*», ont attiré l'attention des chercheurs et industriels de l'ingénierie logicielle. En outre, avec l'évolution de SOA et la maturité progressive de la technologie du Cloud Computing, l'accent est mis sur la promotion de la réutilisation des services SaaS.

Le Cloud computing est essentiellement basée sur trois niveaux d'hébergement d'offre de services :Software as a Service (SaaS), Platform as a Service (PaaS) et Infrastructure as Service (IaaS). On peut identifier six principales caractéristiques d'un service SaaS qui sont la réutilisation, les données gérées par le fournisseur, la personnalisation de service, la disponibilité, l'évolutivité et le paiement à l'utilisation. En outre, l'élasticité rapide et la mise en commun des ressources qui sont des caractéristiques essentielles de Cloud, améliorent la réutilisation et contribuent à Promouvoir la variabilité des services SaaS dans le Cloud computing et en particulier dans des environnements multi-tenants. En effet, la majorité des services Cloud, composants et applications SaaS sont construites avec multi-tenancy. Dans [5], le multi-tenancy est défini comme une architecture logicielle pour les plateformes et les applications servant simultanément multiples clients, tout en garantissant l'accès isolé aux systèmes.

Dans ce contexte, les clients sont identifiés en tant que tenants et sont logiquement séparées, mais physiquement intègres dans une architecture multi-tenant.Dans ce mémoire, nous proposons une approche pour la découverte de services SaaS configurables dans le Cloud computing. Le modèles de caractéristiques ou Feature model est utilisé comme modèle abstrait pour exprimer la variabilité des services SaaS configurables. En outre, un module pour le matching de ces services est développé. Pour cela, le problème de services SaaS configurables est réduit à un problème de matching d'arbres par l'extension d'algorithmes existants à cette fin.

Chapitre 1. **Concepts de bases**

Introduction

Dans ce chapitre, on va présenter les notions de base pour l'architecture SOA et les services web, leur caractéristiques dans le Cloud.

1. Architecture SOA (Service Oriented Architecture)

Un service est une fonction logicielle autonome et sans état qui accepte des requêtes et qui renvoie des réponses au travers d'une interface standard bien défini. Un service est donc une unité de traitement qui fournit un résultat à un consommateur. Fournisseurs et consommateurs sont habituellement des agents logiciels qui agissent par délégation de leurs propriétaires. Les services ne doivent pas dépendre de l'état d'autres fonctions ou d'autres traitements externes. Les technologies employées pour réaliser un service comme le langage de programmation ne font pas partie de la définition d'un service. L'architecture orientée service (*Service Oriented Architecture*) SOA est une architecture logicielle ou les fonctionnalités sont groupées dans des processus métiers et emballées en tant que services.

2. Les services web

2.1. Définition

Un service Web est un programme informatique permettant la communication et l'échange de données entre applications et systèmes hétérogènes dans des environnements distribués. Il s'agit donc d'un ensemble de fonctionnalités exposées sur internet ou sur un intranet, par des applications ou machines, sans intervention humaine, et en temps réel. Techniquement, un Service Web (SW) peut donc être perçu comme étant une interface décrivant une collection d'opérations accessibles via le réseau à travers des messages XML standardisées.

2.2. Architecture

L'architecture de référence des SW (Figure 1) s'articule autour des trois rôles suivants :

- **Fournisseur de service :** correspond au propriétaire du service. D'un point de vue technique, il est constitué par la plate-forme d'accueil du service.

-Client : correspond au demandeur de service. D'un point de vue technique, il est constitue par l'application qui va rechercher et invoquer un service. L'application cliente peut être Elle-même un SW.

- Annuaire des services : correspond à un registre de descriptions de services offrant des facilitées de publication de services à l'intention des fournisseurs ainsi que des facilites de recherche de services à l'intention des clients. Les interactions de base entre ces trois rôles incluent les opérations de publication, de recherche et de liens d'opérations. Nous décrivons ci-dessous un scenario type d'utilisation de cette architecture. Le fournisseur de services définit la description de son service et la publie dans un annuaire de service. Le client utilise les facilitées de recherche disponibles au niveau de l'annuaire pour retrouver et sélectionner un service donné. Il examine ensuite la description du service sélectionne pour récupérer les informations nécessaires lui permettant de se connecter au fournisseur du service et d'interagir avec l'implémentation du service considéré.

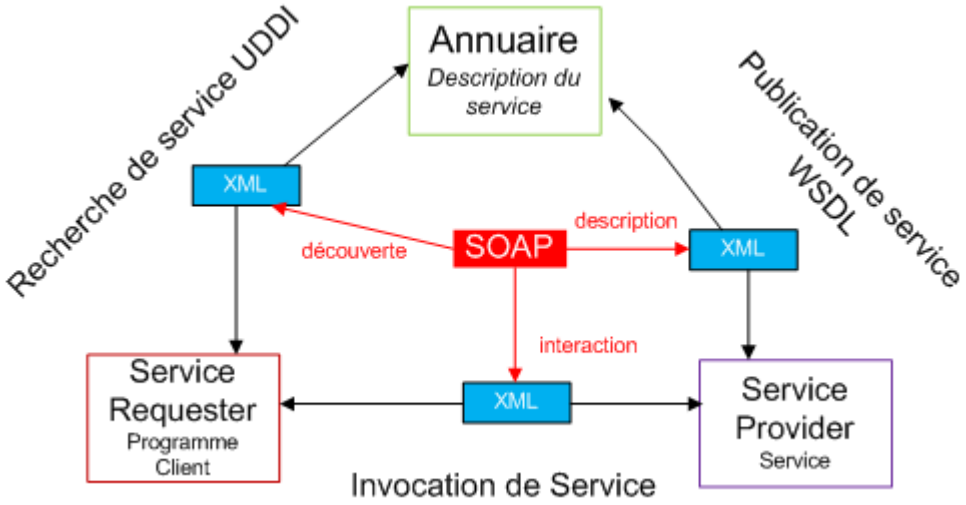

Figure 1: Architecture des services web [1]

2.3. Description des interfaces

La description des interfaces d'un SW est assurée par WSDL (*Web Service Description Languate*) qui a été proposé en 2001 par W3C (*World Wide Web Consortium*) pour standardisation.

Un document WSDL est composé de ces éléments pour définir un service web :

-**Type :** Il s'agit d'un conteneur pour décrire les types de données utilisées dans la description des messages échanges.

- **Message :** C'est une définition abstraite et typique des données échangés.

- **Opération :** Cet élément s'occupe de la description de l'action supporté par le service.

- **Port Type :** Un ensemble d'opérations abstraites supporté par un ou plusieurs endpoint. Chaque opération réfère à des messages input et output.

- **Binding :** Il s'agit d'un protocole concret et d'une spécification de format de données pour un port type particulier.

- **Port :** C'est un endpoint défini par une combinaison des bindings et des adresses réseau.

- **Service :** Le dernier élément "service" définit une collection des endpoints relatifs à ce service.

➔Le diagramme de classe de la figure I.2 est un méta-modèle qui représente les éléments de l'interface d'un SW. Cette interface est définie par les éléments de WSDL (Types, messages, PortTypes...) .

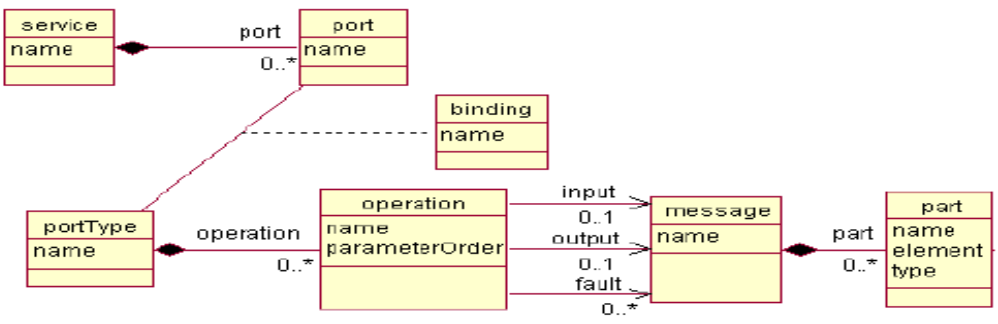

Figure 2:Eléments de WSDL

3. Cloud Computing

3.1. Définition

NIST [10] définit le Cloud Computing comme étant un modèle qui vise à supporter, délivrer et consommer des logiciels et des ressources physiques à la demande à travers les protocoles d'Internet. Effectivement, le Cloud est un empilement de services référant à Infrastructure as a Service , Platform as a Service et Software as a Service.

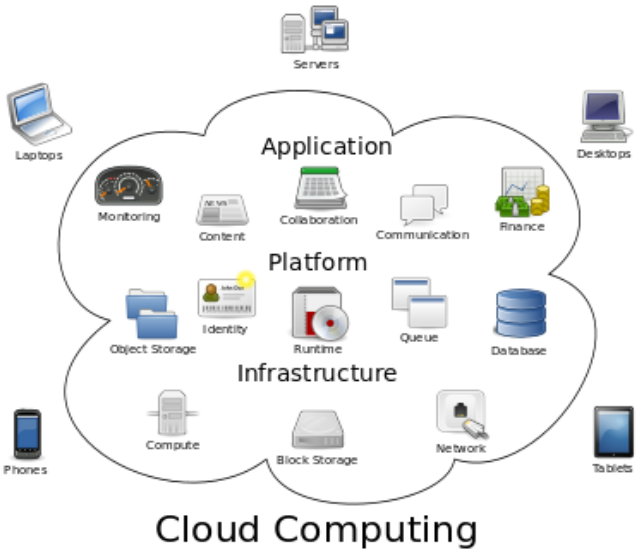

Figure 3:Cloud Computing [1]

3.2. Caractéristiques

Le Cloud computing est composée de sept caractéristiques essentielles :

3.2.1. Multi-tenancy

Le (*Multi-tenancy*) ou (*Multi-location*) est une notion qui est apparue avec le SaaS. Le terme multi-tenancy désigne une architecture dans laquelle une seule instance d'une application logicielle serve plusieurs clients. Chaque client est appelle un tenant.

3.2.2. Services à la demande

C'est la capacité de consommer automatiquement des ressources informatique telles que la machine, le stockage réseau, en fonction des besoins, sans l'intervention humaine avec le fournisseur de service.

3.2.3. Accès au réseau

Cette caractéristique permet la disponibilité et l'accès au réseau en utilisant un navigateur, un client lourd ou tout autre appareil (Smartphone, tablette).

3.2.4. Virtualisation

La virtualisation est une caractéristique indispensable qui présente de très nombreux avantages.Le matériel est remplacé par du logiciel avec tous les avantages du logiciel. La virtualisation est un moyen de maximiser l'utilisation des ressources matérielles, des systèmes d'exploitation et des applications, car elle simule des ressources qui sont pas présents physiquement.

3.2.5. Elasticité rapide

C'est la capacité d'une application de passer à l'échelle de manière dynamique et rapide en fonction des besoins. Du point de vue consommateur, les ressources utilisées semble être illimitées et peuvent être louées à n'importe quel moment et en quantité illimite.

3.3. Modèles de services Cloud

3.3.1. Software as a Service SaaS

Le Software as a Service est un nouveau modèle émergent de conception, implémentation et de déploiement de logiciels. Les fournisseurs de logiciels possèdent le logiciel et libèrent ses fonctions à un ensemble de Clients sur Internet. Ces clients utilisent le logiciel à travers l'interface API accessible par le web. D'où les applications logicielles sont fournis aux clients comme étant un service et pas un produit.

3.3.2. Plateform as a Service PaaS

Platform as a Service, fournit des plateformes informatiques qui englobent généralement le système d'exploitation, la programmation de l'environnement d'exécution du langage, base de données, serveur web, etc. PaaS est une solution très évolutive, et les utilisateurs n'ont pas à se soucier des mises à niveau de la plateforme ou de problèmes d'indisponibilité pendant la maintenance.

3.3.3. Infrastructure as a Service IaaS

Infrastructure as a service, fournit l'infrastructure informatique, les machines virtuelles ou physiques et d'autres ressources comme les serveurs de fichiers, le stockage des données, Pare-feu, les équilibreurs de charge, les adresses IP, les réseaux locaux virtuels, etc. . . .

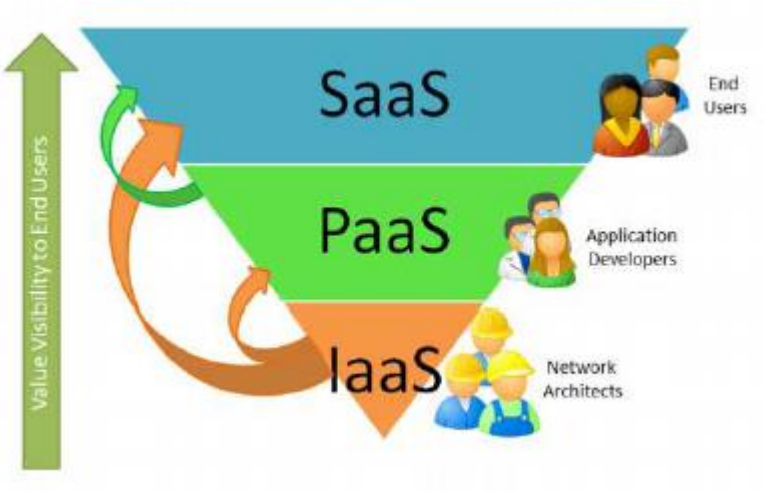

Figure 4:Différents modèles de services Cloud [3]

4. Caractéristiques des services SaaS

4.1. Variabilité

La variabilité se définit en général par l'aptitude d'un système à s'adapter, à se spécialiser et à se configurer en fonction du contexte de son utilisation. La variabilité a fait l'objet de plusieurs applications dans plusieurs domaines, tels que l'ingénierie des domaines, des lignes de produits, des composants, des services, etc. Selon le domaine d'application, ce concept peut faire l'objet de différentes interprétations.

Dans l'ingénierie de services, la variabilité est définie comme une technique centrale permettant la réutilisation de services dans différents scenario d'exécution ainsi que la simplification de leur utilisation par les consommateurs. Dans [4], les auteurs donnent une présentation des propriétés de la variabilité. Il s'agit de l'ensemble des éléments nécessaires à la représentation de la variabilité dans les artefacts.

Parmi les propriétés les plus utilisées dans la littérature, nous citons : Unités de la variabilité, Portée de la variabilité et types de la variabilité.

➔***Unités de variabilité :*** Sont définies initialement dans les lignes de produits. Deux concepts sont employés :

(a) **Point de variation** : c'est une partie du système ou des choix doivent être faits afin d'identifier les variantes à utiliser.

(b) **Variante** : représente une réalisation spécifique d'un point de variation.

➔***Portée de la variabilité :*** elle signifie les contraintes fournies par le point de variation sur le choix de ses variantes. Selon [5], il existe 4 portées :

(a) **Optionnelle :** les variantes qui peuvent être sélectionnées ou non.

(b) **Alternative :** permet le choix d'une seule variante parmi n choix possibles (XOR).

(c) **Alternative optionnelle :** il s'agit d'un point de variation de portée optionnelle qui offre des variantes alternatives. Ce type de variation permet le choix d'aucune ou d'une seule variante parmi les n choix possibles.

(d) **Ensemble d'alternatives :** cela signifie qu'il existe de multiples réalisations de cette variation et qu'au moins une doit être sélectionnée.

➔***Types de variabilité :*** ceci permet de classifier les points de variation selon la dimension, la visibilité ou le niveau de représentation.

(a) ***selon la dimension,*** il y a 2 types : variabilité dans le temps (différents versions d'un artefact à différents moments) et variabilité dans l'espace (un artefact capture différentes formes en même temps).

(b) ***Selon la visibilité,*** on peut classifier la variabilité en interne, ou les variantes sont visibles seulement au développeur et non à l'utilisateur, et en externe ou les variante sont visibles aux deux acteurs.

4.2. Configurabilité

Dans [6], les auteurs introduisent la configurabilité des services SaaS qui vise à assurer aux tenants une multitude d'options et de variations en utilisant un seul code de base, de sorte qu'il est possible pour chaque tenant d'avoir une unique configuration du logiciel. La configurabilité doit être facile et intuitive pour les concepteurs et doit satisfaire les besoinsdes utilisateurs. Chaque tenant a besoin des données de configuration. C'est pourquoi, pour un tenant, il existe deux types d'utilisateurs : un concepteur et un utilisateur. Le concepteur crée les données de configuration et l'utilisateur consomme ces données pour chaque application SaaS. Les données de configuration jouent un rôle très important dans la réalisation de la configurabilité des applications SaaS. Mais, à part ces données, il y a les données spécifiques de l'application qui sont disponibles à l'utilisateur lors de l'exécution. Notons bien que les données de l'application sont séparées des données de configuration.

5. Problématique

Un Service web classique est une entité autonome et modulaire de logiciels fournis par un fournisseur de services pour remplir les fonctionnalités désirées pour un consommateur.

Aussi, les services Web sont généralement utiles pour favoriser la réutilisation des logiciels pour créer de nouvelles applications non à partir de zéro mais en réutilisant ceux qui existent déjà, soit par extension ou médication. Le processus de découverte de service Web joue un rôle fondamental pour faire le lien entre des informations publiées par des fournisseurs et des requêtes crées par les internautes. Comme W3C (Booth et al., 2004) a définit, la découverte est considère comme l'acte de localiser une description d'un service Web par une machine qui répond à certains critères fonctionnels.

En conséquence à l'investissement excessif dans la gestion des services, le nombre de services publies sur internet ne cesse pas à se multiplier. Par conséquent, cette croissance entrave la découverte efficace d'un service spécifique au sein des registres de taille importante.

Généralement, la découverte de service Web est basée sur des techniques de matching pour trouver des services en comparant leurs descriptions vis à vis les contraintes et les préférences des consommateurs

Conclusion

Après avoir définit les concepts de base de notre sujet de recherche (le Cloud Computing, les services web...), on a présenté la problématique. Dans le chapitre qui suit, nous présentons l'état de l'art.

Chapitre 2. **Etat de l'art**

Introduction

Dans ce chapitre, on va présenter les modèles utilisés pour la modélisation de la variabilité ainsi que les différents travaux antérieurs de découverte de services SaaS.

1. Modèles de variabilité des lignes de produits

1.1. Feature model

En développement logiciel, les **Feature model** sont des techniques de modélisation des Software product line. Ils ont été introduits pour la première fois dans la méthode de feature-oriented domain analysis par Kang en 1990[11].Les feature models sont utilisés pour identifier les différences et similitudes entre tous les produits d'une même software product line. Chaque caractéristique, ou feature, peut être un préalable, un composant ou un morceau de code.

Les feature model se présentent commet des arbres dans lesquels chaque nœud est une caractéristique et chaque arête peut avoir quatre valeurs possibles :

- Obligatoire : Si le nœud parent est présent, alors le nœud enfant aussi.
- Optionnel : Le nœud enfant peut ou peut ne pas être présent alors que le nœud parent l'est.
- Alternative : Si le nœud parent est présent, alors un seul des nœuds enfants reliés par une arête alternative l'est.
- Ou : Si le nœud parent est présent, alors au moins un des nœuds enfants reliés par une arête l'est.

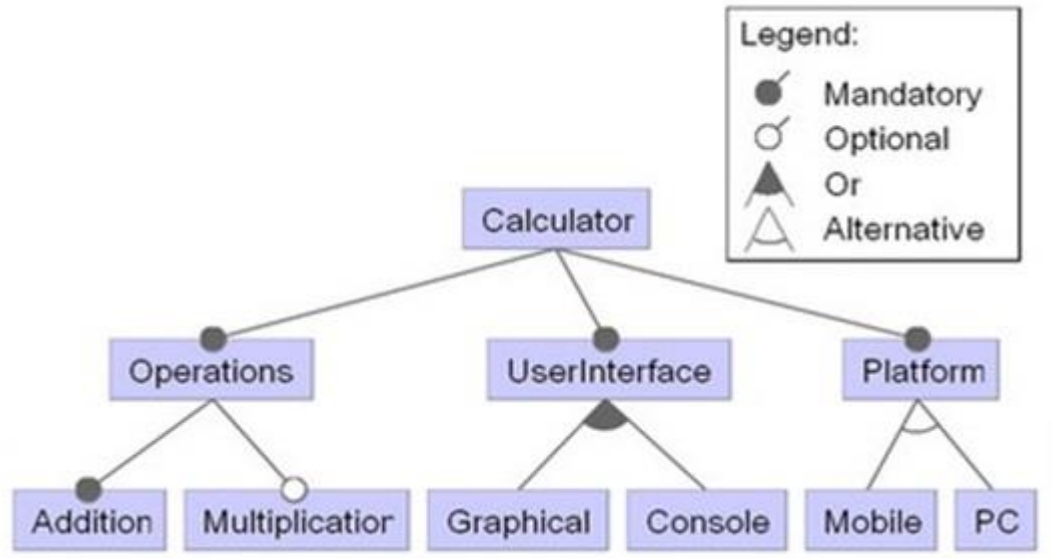

Figure 5:Exemple de FM

1.2. Orthogonal Variability Model

(*Orthogonal Variability Model*) OVM est un modèle de variabilité propose par Klaus Pohl [7] pour la gestion de la variabilité dans les applications, en terme d'exigences, architecture, composants et test d'artefacts. Dans ce modèle, seulement la variabilité de la ligne de produit est documentée.

Un point de variation (VP) documente un élément de variabilité et une variante (V) documente les instances possibles de cet élément de variabilité.

Les points de variation et les variantes peuvent être soit optionnels soit obligatoires. Un point de variation obligatoire doit être toujours lié, c'est-à-dire, tous les produits de cette ligne de produit doivent avoir ce point de variation (VP). Un VP optionnel ne doit pas être liée, il peut être choisi seulement pour un produit spécifique. Pour un VP quel que soit obligatoire ou optionnel, ses variantes obligatoires doivent être choisis, et les optionnels peuvent l'être, mais pas forcément. Dans OVM, les variables optionnels peuvent être regroupés en choix alternatifs avec une cardinalité [min...Max]. Toutes ces notations sont illustrées dans la figure ci-dessous.

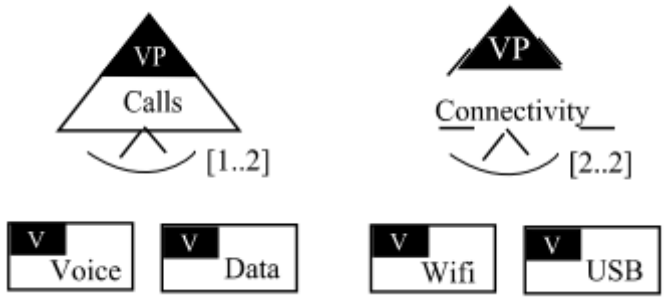

Figure 6:Notation graphique de OVM [8]

2. Algorithmes de matching d'arbres

2.1. Algorithme Simple Tree Matching

Dans l'article de (Wang et al., 2010) , les deux chercheurs, et en se basant sur l'algorithme (*Simple Tree Matching*) STM, ont présente une méthode d'extraction des données web par l'analyse du structure des documents web.

Nous nous intéressons à l'adaptation de l'algorithme de matching d'arbres pour comparer et mesurer la similarité de deux services SaaS configurables (cotée fournisseur et client), pour pouvoir sélectionner le service SaaS le plus proche aux exigences du client. Le principe de fonctionnement de l'algorithme STM est comme suit :

Algorithm Simple Tree Matching (A,B)

1: if the roots of the two trees A and B contains distinct symbols then
2: Return 0 ;
3: else m = the number of first level sub trees of A;
4: n = the number of first level sub trees of B;
5: Initialization : M[i,0] for i=0,..,m;
6: M[0,j] for j=0,..,m;
7: for i = 1 to m do
8: for j = 1 to n do
9: M[i, j] = max(M[i, j-1], M[i-1, j], M[i-1, j-1] + W[i, j]) where W[i, j] = SimpleTreeMatching(Ai,Bj) ;
10: end for
11: end for
12: RETURN (M[m][n] + 1); 13: end if.

A et B sont les deux arbres à comparer, i et j représentent deux nœuds dans A et B respectivement. La valeur du matching maximum entre deux arbres est le matching qui a le nombre maximum de paires en correspondance. On considère A= RA :<A1, . . . ,Am> et B= RB :<B1,. . . .Bn> sont deux arbres où RA et RB sont les racines de A et B respectivement, Ai et Bj sont les ième et jème nœuds des sous-arbres du premier niveau des arbres A et B respectivement, et m et n représentent la taille des deux arbres. Si le contenu des nœuds RA et RB est le même, alors le maximum matching (i.e.W(A,B)) est M(<A1 ,A2 , . . . ,Am>,<B1, B2, . . . ,Bn>)+1 Ou M(<A1,A2..,Am>,<B1, B2, . . .,Bn>)= max (M(<A1 ,A2 , . . .,Am-1>,<B1, B2, . . . ,Bn-1>)+W(Am ,Bn), M (<A1 ,A2 , . ..,Am-1>,<B1, B2, . . . ,Bn>), M (<A1 ,A2 , ...,Am>,<B1, B2, . . . ,Bn-1>)) .Lorsque RA et RB ont un contenu distinct, alors W (A, B)=0.Vue que Simple Tree Matching calcule la similarité en utilisant la programmation dynamique pour générer le plus grand matching, la complexité de cet algorithme est de O(mn).En premier lieu, l'algorithme compare les racines des arbres. S'il trouve qu'elles sont distinctes, les deux arbres ne correspondent pas, sinon, il continue d'une façon récursive à chercher le maximum matching entre les deux sous arbres du premier niveau des arbres en cours. La valeur du matching maximum calculée est stockée dans la matrice W, et à partir de cette valeur, on calcule la valeur de la matrice M. Les premières lignes et colonnes de la matrice M sont initialisées à 0. Les avantages de l'adoption de cet algorithme, qui a été montré très efficace pour l'extraction de données Web, sont multiples ; par exemple, STM évalue la similarité entre deux arbres en produisant le matching maximum grâce à une programmation dynamique, sans calculer l'insertion, la modification et la suppression des opérations. En outre, les algorithmes de calcul de distance d'édition d'arbres repose sur des implémentations complexes pour obtenir de bonnes performances, alors que STM est simple à implémenter. Il y a quelques limites ; la plupart d'entre eux sont sans importance, mais il existe une importante : cette approche ne peut pas correspondre à des permutations de nœuds. Après, pour obtenir la valeur de similarité entre les deux arbres, et qui est comprise dans l'intervalle [0..1] on utilise la formule suivante :

Similarity (A,B) =Simple Tree Matching(A,B)/((*Size(A) + Size(B)/2)* Ou : Simple Tree Matching (A, B) est la valeur retournée par l'algorithme. *Size(A)* est le nombre de nœuds de l'arbre A et *Size(B)* le nombre de nœuds de l'arbre B Plus la valeur obtenue est proche à 1 plus les arbres sont similaires.

Trace d'exécution de l'algorithme STM :

On prend l'exemple des deux arbres A et B suivants pour calculer la valeur de matching maximum entre eux puis la similarité :

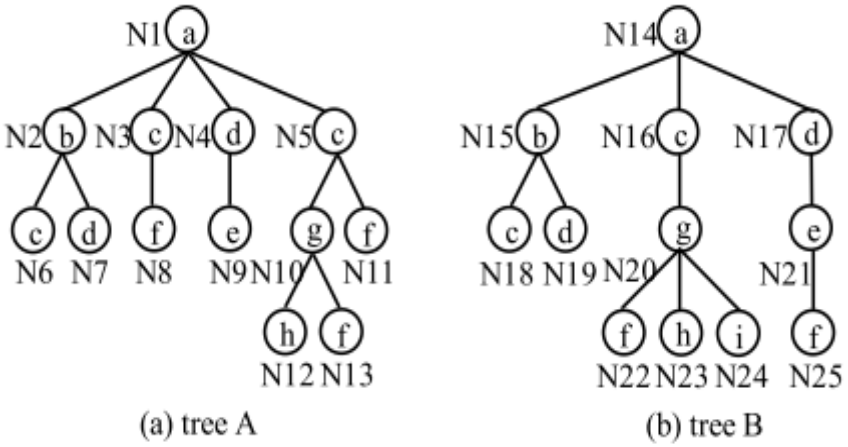

Figure 7: Deux arbres A et B

Les matrices M et W de cet exemple sont présentées ci dessous :

Tableau 1:M 1, 14

	0	N15	N15-N16	N15-N17
0	0	0	0	0
N2	0	3	3	3
N2-N3	0	3	4	4
N2-N4	0	3	4	6
N2-N5	0	3	6	6

Tableau 2:W 1, 14

	N15	N16	N17
N2	3	0	0
N3	0	1	0
N4	0	0	2
N5	0	3	0

Tableau 3:M 2, 15

	0	N18	N18-N19
0	0	0	0
N6	0	1	1
N6-N7	0	1	2

Tableau 4: W2, 15

	N18	N19
N6	1	0
N7	0	1

Tableau 5:M 3, 16

	0	N20
0	0	0
N8	0	0

Tableau 6:W 3, 16

	N20
N8	0

Tableau 7:M 4, 17

	0	N21
0	0	0
N9	0	0

Tableau 8:W 4, 17

	N21
N9	1

Tableau 9:M 5, 16

	0	N20
0	0	0
N10	0	2
N10-N11	0	2

Tableau 10:W 5, 16

	N20
N10	2
N11	0

Tableau 11:M 10, 20

	0	N22	N22-N23	N22-N24
0	0	0	0	0
N12	0	0	1	1

Tableau 12:W 10,20

	N22	N23	N24
N12	0	1	0
N13	1	0	0

2.2. Algorithme Clustered Tree Matching

L'algorithme Clustered Tree Matching CTM est un autre algorithme de matching d'arbres. Il se base sur le principe de STM.

Cet algorithme est proposé par (Ferrara et al., 2011) . Ils ont appliquée quelques modifications par rapport à Simple Tree Matching au niveau de l'attribution d'une valeur pour chaque nœud en correspondance.

Dans STM la valeur affectée est toujours 1. Dans le nouvel algorithme, le principe est de ne pas donner beaucoup d'importance aux petites différences de structure des deux arbres situées dans un niveau profond (nœuds feuilles) et aussi lorsqu'il s'agit de niveaux contenant plusieurs nœuds. Donc, la valeur devient 1 divisée par le nombre le plus grand de «Sibling nodes» des deux nœuds à comparer. Cette division permet de réduire l'impact de plusieurs nœuds dans un même niveau. La différence entre les deux algorithmes se voit dans la ligne 12 où on va la changer par le code suivant :

Algorithm Clustered Tree Matching (A,B)

1: if m > 0 AND n > 0 then
2: RETURN M[m][n] * 1 / Max(t(A),t(B))
3: else
4: RETURN M[m][n]+ 1 / Max(t(A),t(B))
5: end if

Où t(X) représente le nombre de «sibling nodes» du nœud X y compris lui-même, et t(B) pour le nœud B aussi la même chose.

Trace d'exécution de l'algorithme CTM :

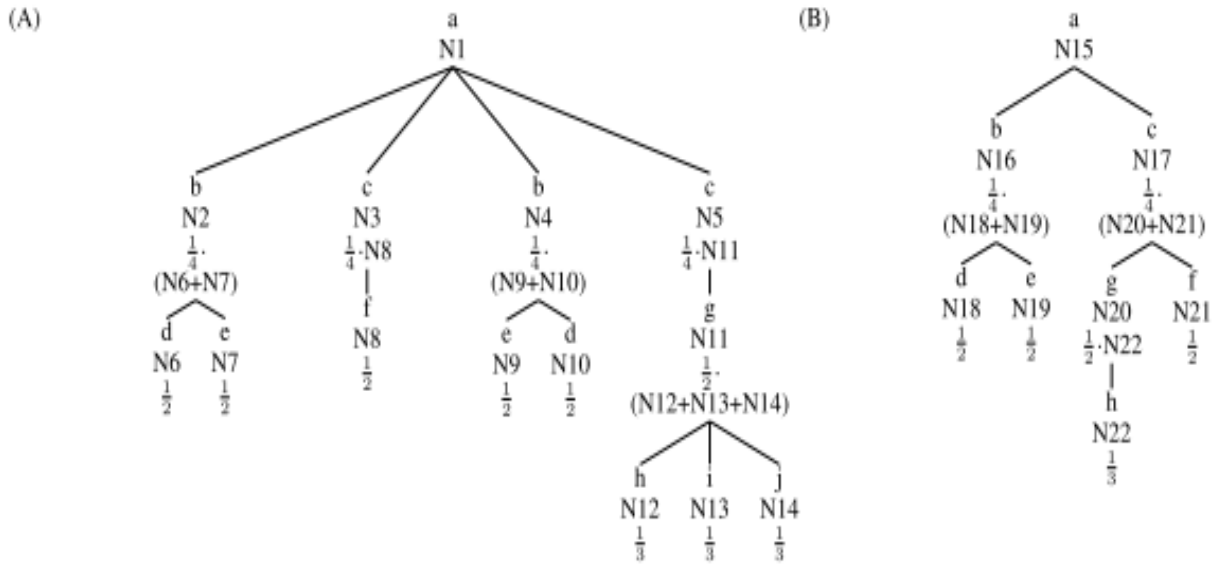

Figure 8: Deux arbres A et B

Tableau 13:W 2,16

	0	N18	N18-N19
0	0	0	0
N6	0	1/2	1/2
N6-N7	0	1/2	1

Tableau 14:M 2, 16

	N18	N19
N6	1/2	0
N7	0	1/2

Tableau 15:M 4, 16

	0	N18	N18-N19
0	0	0	0
N9	0	0	1/2
N9-N10	0	1/2	1/2

Tableau 16:W 4, 16

	N18	N19
N9	1/2	0
N10	0	1/2

Tableau 17:M 3, 17

	0	N8
0	0	0
N20	0	0
N20-N21	0	1/2

Tableau 18: W 3, 17

	N8
N20	0
N21	1/2

Tableau 19: M 11, 20

	0	N12	N12-N13	N12-N14
0	0	0	0	0
N22	0	1/3	1/3	1/3

Tableau 20: M 11, 20

	N12	N13	N14
N22	1/3	0	0

Tableau 21:M 5, 17

	0	N11
0	0	0
N20	0	1/6
N20-N21	0	1/6

Tableau 22:W5, 17

	N11
N20	1/6
N21	0

Tableau 23:M 1, 15

	0	N2	N2-N3	N2-N4	N2-N5
0	0	0	0	0	0
N16	0	1/4	1/4	1/4	1/4
N16 - N17	0	1/4	3/8	3/8	3/8

Tableau 24:W 1, 15

	N2	N3	N4	N5
16	1/4	0	1/8	0
17	0	1/8	0	1/24

2.3. Algorithm Measuring Similarity with Matrix

Dans l'article de (*Israt J. Chowdhury et Richi Nayak 2013*), les deux chercheurs, et en se basant sur l'algorithme (*Measuring Similarity with Matrix*), ont présente une nouvelle méthode pour trouver efficacement les similitudes entre les arbres désordonnées. Il consiste d'abord à encoder un arbre avec une approche optimale de parcours, pour modéliser par la suite l'arbre avec sa représentation matricielle équivalente. L'analyse empirique montre que la méthode proposée est capable d'atteindre une grande précision, même sur les grands ensembles de données.

Nous nous intéressons à l'adaptation de l'algorithme de Similarité pour comparer et mesurer la similarité de deux services SaaS configurables (cotée fournisseur et client), pour pouvoir sélectionner le service web le plus proche aux exigences du client. Le principe de fonctionnement de l'algorithme est comme suit :

Algorithm Measuring Similarity with Matrix (A, B)

Input: Unordered trees *Ta* and *Tb*

Output: Measurement similarity between tree pair

1. Model the tree *Ta* with the Augmented Adjacency Matrix *A'*;
2. Model the tree *Tb* with the Augmented Adjacency Matrix *B'*;
3. **if** $|B'|>|A'|$ **then**

Add ($|B'| - |A'|$) rows and columns of zeros at the right end and bottom of the matrix *A'*;

else

Add ($|A'| - |B'|$) rows and columns of zeros at the right end and bottom of the matrix *B'*;

end if

4. Calculate similarity between two trees using

$$Cos(A',B') = \frac{\sum_{x=1}^{n}\sum_{y=1}^{n} A'_{xy} B'_{xy}}{\sqrt{\sum_{x=1}^{n}\sum_{y=1}^{n} A'^{2}_{xy}} \sqrt{\sum_{x=1}^{n}\sum_{y=1}^{n} B'^{2}_{xy}}}$$

La méthode *Measuring Similarity with Matrix* proposée comprend trois étapes. **Tout d'abord**, les données de l'arbre sont codées avec une approche optimale de déplacement. **D'autre part**, une représentation matricielle équivalente est obtenue pour chaque structure arborescente en utilisant le codage de l'arbre avec d'autres informations de l'arborescence. **En troisième lieu**, la mesure de similarité cosinus est utilisée pour calculer la similarité entre deux matrices représentant des arbres non ordonnées.

Etape 1: le codage de l'arbre à l'aide d'une approche optimale de parcours :

-Parcours d'Arbre : Un parcours d'arbre est une approche systématique de visiter chaque nœud une fois dans un arbre en suivant certaine stratégie et retourne une liste contenant la séquence de nœud traversé le long du chemin. La recherche en profondeur d'abord **(DFS)** et la recherche en largeur d'abord **(BFS)** sont deux algorithmes traversant couramment utilisés qui dépendent de l'ordre fixe entre les nœuds frères et sœurs. Un algorithme DFS commence à partir du nœud racine et explore chaque branche, autant que possible avant de revenir en arrière. Ils peuvent être classés comme pre-order, in-order et post-order, sur la base de la séquence de l'apparition des nœuds sur commande droite ou à gauche. Un algorithme BFS, aussi connu comme l'ordre de niveau algorithme de parcours, commence la visite d'un arbre de son nœud racine, puis suit une stratégie pour traverser d'autres nœuds dans l'ordre de leur niveau de gauche à droite.

-Parcours Optimale d'un Arbre: Cette méthode est inspirée par un problème d'optimisation bien connue appelée "*Simple Assemble line balancing*" du paradigme "recherche opérationnelle".

-Encodage d'Arbre : Après réception de la séquence optimale pour traverser tous les nœuds de l'arborescence, chaque nœud est codé en fonction de son ordre dans cette séquence. Par exemple, sur la Figure 9, le parcours démarre à partir du nœud racine *Va* et la séquence optimale est *Va-Vc-Ve- Vb-Vf-Vd*. Les valeurs codées pour les nœuds de l'arbre seront 1-2-3-4-5-6 pour Va-VC- Ve-Vs-Vf-Vd respectivement.

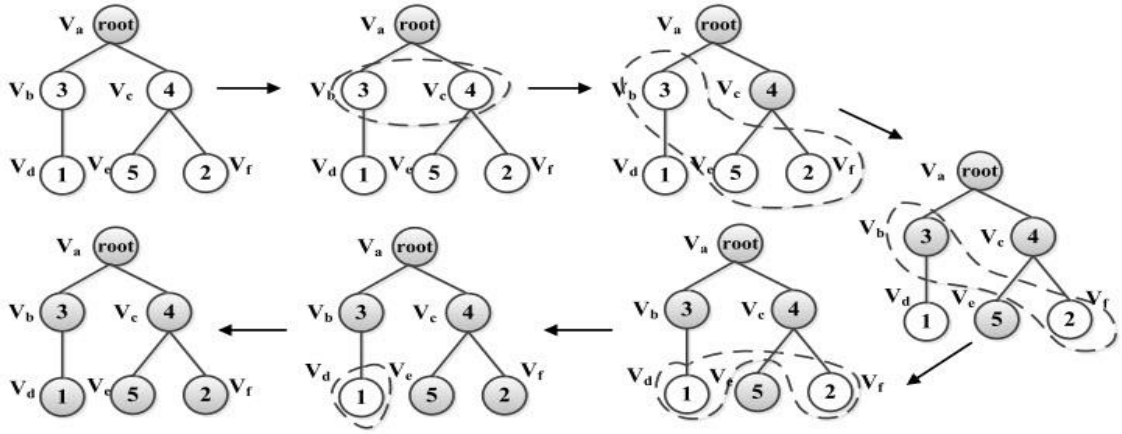

Figure 9:Encodage d'arbre

Etape 2: Modélisation d'Arbre avec la Représentation de Matrice d'adjacence augmentée:

-Matrice d'adjacence augmentée : Ceci est une matrice carrée qui utilise le niveau, le codage et le poids des informations d'un arbre pour représenter les valeurs de cellule.

-Encodage informations: En utilisant la séquence de Parcours Optimale d'un Arbre, on obtient les valeurs de codage des nœuds d'arbre selon l'ordre dans lequel ils sont visités. Le nœud racine devient la première ligne et la colonne d'être représentés dans la matrice et les autres nœuds sont disposés dans l'ordre optimal obtenu par la **Parcours optimale**. Cette valeur de codage intègre également avec la valeur de niveau entre deux nœuds.

-L'information de niveau: L'information de niveau dans un arbre représente les relations d'ascendance des nœuds.

1. Si une relation ancêtre-descendant existe entre deux nœuds *Vi* et *Vj*, où *Vi* est l'ancêtre de *Vj*, ou si la valeur de codage de *Vi* est inférieure à la valeur de codage de *Vj* alors la valeur du niveau de la cellule *Cij* est: niveau *(Vj) /* niveau *(Vi)*
2. Si une relation ancestrale n'existe pas entre deux nœuds Vi et Vj, ou si la valeur de codage de Vi est supérieure à la valeur de codage de Vj alors la valeur de niveau pour la cellule Cij sera 0.

- Informations de poids: les nœuds portent un poids affiché la fréquence à laquelle le nœud se produit dans le nœud parent. Le poids de nœud est ajouté à la valeur du niveau correspondant. En outre, une valeur 1 est ajoutée à chaque cellule diagonale de la matrice d'adjacence pour représenter l'existence d'un nœud correspondant à cet arbre.

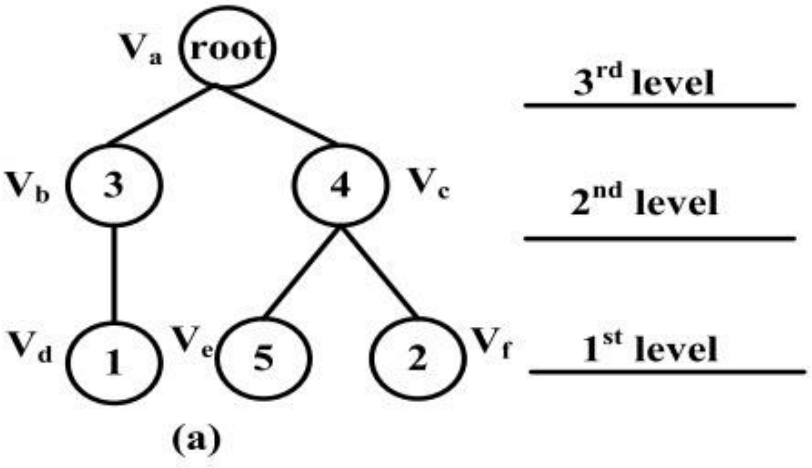

Figure 10:Les niveaux d'arbre

Tableau 25:Matrice d'adjacence traditionnelle

	Va(1)	Vc(2)	Ve(3)	Vb(4)	Vf(5)	Vd(6)
Va(1)	0	1	0	1	0	0
Vc(2)	0	0	1	0	1	0
Ve(3)	0	0	0	0	0	0
Vb(4)	0	0	0	0	0	1
Vf(5)	0	0	0	0	0	0
Vd(6)	0	0	0	0	0	0

Tableau 26:Matrice d'adjacence augmentée

	Va(1)	Vc(2)	Ve(4)	Vb(4)	Vf(5)	Vd(6)
Va(1)	1	2/3+4	1/3	2/3+3	1/3	1/3
Vc(2)	0	1	1/2+5	0	1/2+2	0
Ve(3)	0	0	1	0	0	0
Vb(4)	0	0	0	1	0	1/2+1
Vf(5)	0	0	0	0	1	0
Vd(6)	0	0	0	0	0	1

Nous illustrons le processus de modélisation de l'arbre avec la matrice d'adjacence augmentée et remplir les valeurs de la matrice. La figure 10 illustre la matrice d'adjacence traditionnelle et la matrice de contiguïté augmentée pour un arbre donné. L'exemple de l'arbre possède trois niveaux, et le niveau de nœud racine est considéré comme le plus élevé. La valeur de codage de nœuds est reçue de la figure 9 en utilisant le parcours optimal. La séquence de parcours est *Va-Vc-Ve-Vb-Vf-Vd* et les valeurs de codage pour ces nœuds sont 1-2-3-4-5-6 respectivement. Les informations de niveau de nœuds correspondant sont calculées et les coefficients de pondération de nœud sont ajoutés aux valeurs de niveau. Par exemple, considérons le calcul de la valeur de la cellule, *C23*, montrant la relation entre *Va-Vc*. La valeur de codage de *Va* = 1 qui est inférieure à la valeur de codage de *Vc* = 2 signifie que Va est l'ancêtre de *Vc*. Selon la règle 1, la valeur du niveau de *C23* est niveau V (c) / niveau V (c) =2/3. Le poids de Vc est égal à 4. La valeur de la cellule finale sera de 2/3 + 4. Le reste des valeurs de cellules sont en cours calculée de la même manière.

Etape 3: Mesure de similarité :

Soit A 'et B' représentent deux Matrices d'adjacences augmentées des arbres correspondants. Si les deux arbres de tailles différentes, on ajoute à la matrice inférieure des colonnes et des lignes supplémentaires avec des zéro pour rendre la taille des deux arbres égales. Une matrice peut être considérée comme un vecteur à n × n dimensions. La valeur de chaque cellule d'une matrice est une dimension du vecteur, à partir de la première ligne à la ligne d'extrémité; n × n vecteur de dimension est représentée. Similitude entre deux matrices peut être calculée en utilisant la similarité cosinus.

$$Cos(A',B') = \frac{\sum_{x=1}^{n}\sum_{y=1}^{n} A'_{xy} B'_{xy}}{\sqrt{\sum_{x=1}^{n}\sum_{y=1}^{n} A'^{2}_{xy}} \sqrt{\sum_{x=1}^{n}\sum_{y=1}^{n} B'^{2}_{xy}}}$$

3. Comparaison des algorithmes de Matching d'arbres

Soit les deux arbres A et B suivant :

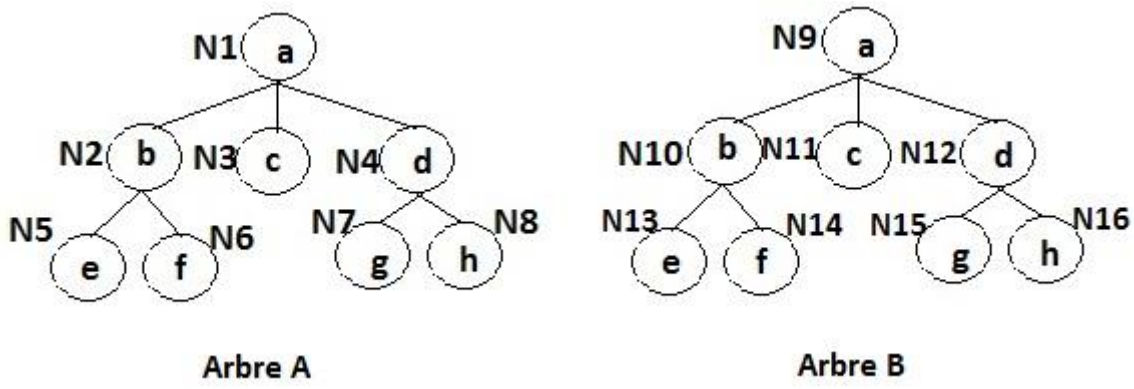

Figure 11: Les deux arbres A et B

En appliquant les trois algorithmes avec ces arbres, on obtient une valeur de similarité égale 1 avec les trois algorithmes car les deux arbres sont identiques. Mais le temps de réponse atteint 6 ms pour STM, 4 ms pour CTM et 80 ms pour SWM.

➔Donc on remarque que le temps de réponse de CTM est plus rapide que le STM et SWM.

Tableau 27:Tableau comparatif de trois algorithmes

	STM	**CTM**	**SWM**
Temps d'exécution	**6 ms**	**4 ms**	**80ms**
Valeur de similarité	**1**	**1**	**1**

Conclusion

Dans ce chapitre, nous avons présentée les travaux récents autour de la découverte de services et applications SaaS configurables. En se basant sur les algorithmes de matching vues dans ce chapitre, nous présentons notre approche de découverte des services SaaS dans le chapitre suivant.

Chapitre 3. **Approche de découverte de services configurables SaaS par matching**

Introduction

Dans ce chapitre, nous allons d'abord donner un aperçu sur notre approche de découverte de services SaaS sensible au multi-tenancy. Ensuite, nous allons illustrer notre approche en présentant une étude de cas. Finalement, des expérimentations quantitatives et qualitatives sont réalisées pour évaluer les deux algorithmes proposées.

1. Aperçu de l'approche de découverte sensible au multi-tenancy

Un aperçu général de notre approche proposée est illustrée par la Figure12 .Les services SaaS configurables sont crées et publiés par les fournisseurs dans les registres de services hébergés dans les centres de données Cloud. La variabilité de ses services est spécifiée à l'aide du modèle de service à base de Feature model, y compris à la fois les caractéristiques fonctionnelles et non fonctionnelles. Chaque tenant soumet une requête de recherche pour le service SaaS qu'il préfère consommer dans le registre multi-tenant de services. Il formule sa requêté en sélectionnant un ensemble de features de services représentant des besoins fonctionnels et des contraintes de qualité comme préoccupations non fonctionnelles pour le service attendu. La suite, un programme de matching est exécuté pour découvrir et localiser les services SaaS candidats. Les instances de services retournées correspondantes à une configuration spécifique de la requête sont uniquement celles qui sont conformes aux contraintes d'attributs exigées par le tenant. De même, les tenants peuvent spécifier les contraintes de déploiement de leurs instances sélectionnées pour décider si ou même avec quels autres locataires ils veulent partager l'instance de service.

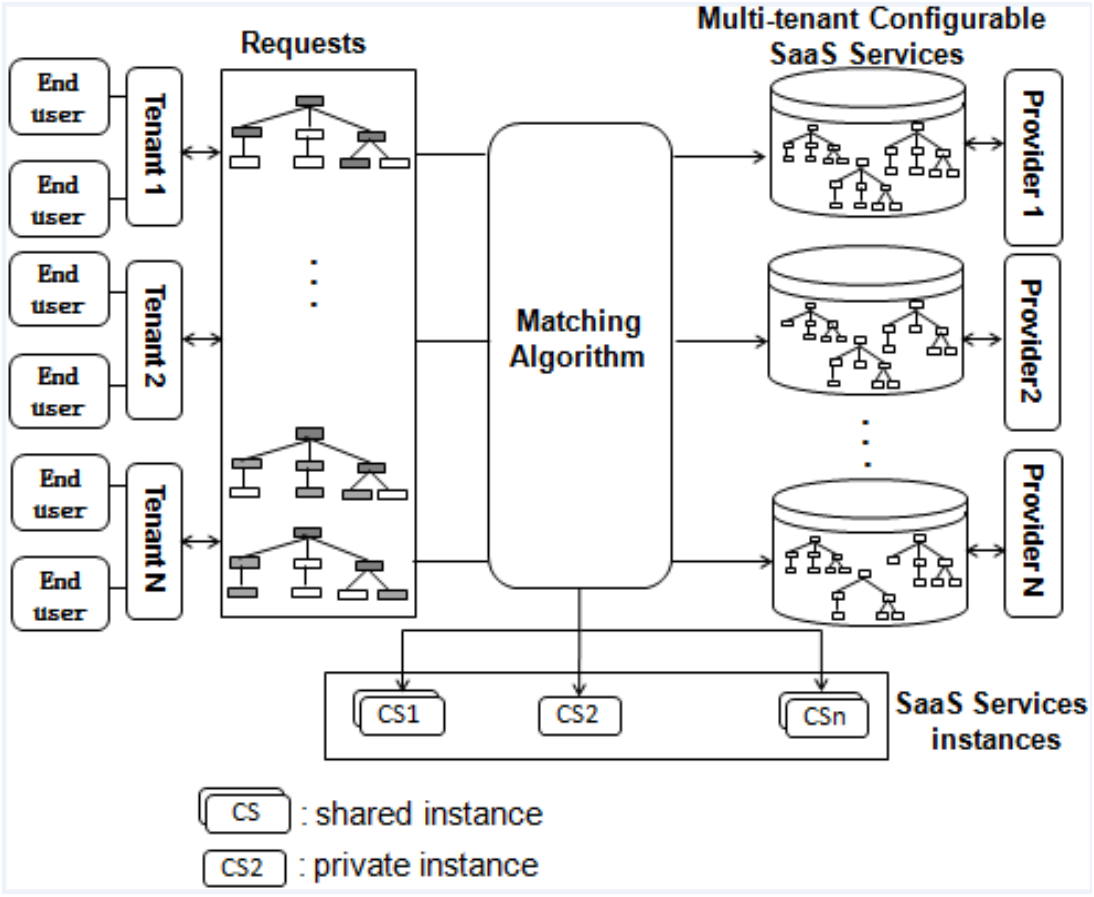

Figure 12:Description du module de découverte

2. Formulation des algorithmes de matching

Dans cette partie, on va présenter notre contribution de matching de services SaaS configurables par extension des deux algorithmes STM, CTM et SWM vues dans le chapitre précédent.

2.1. Algorithme Simple Service Tree Matching

Le principe de fonctionnement de cet algorithme est de calculer la valeur de matching entre deux services SaaS configurables, suivant non seulement les labels de leurs nœuds, mais aussi les valeurs de leurs operateurs. Dans la structure d'un service SaaS configurable basée FM, il existe le nœud racine, des nœuds internes et des nœuds feuilles. Les features dans les nœuds feuilles sont considérées comme nœuds ayant l'operateur mandatory ou optional que nous notons type1. Les features du nœud racine peuvent avoir soit l'operateur or ou alternative que nous notons type2. Les nœuds intermédiaires peuvent avoir les deux types d'operateurs type1 et type2. Nous avons étendu le STM d'origine en ajoutant des conditions de précision sur les nœuds à mapper (à savoir root, interne ou nœud feuille) dans les deux arbres à matcher. Ensuite, nous attribuons une valeur de matching qui considère le cout attribue selon

les types d'operateurs (à savoir, type1, type2). Nous notons cost1 et cost2 les coûts affecté lorsque les valeurs des opérateurs type1 ou type2 sont respectivement distinctes. Si elles sont identiques, la similitude est égal à 1. La contribution dans l'algorithme se voit dans la ligne 12 où on a ajoutée des conditions pour connaitre on est à quel niveau dans les deux arbres à comparer. Puis, on affecte une valeur de matching qui contient en plus les coûts attribuées suivant le type des opérateurs des deux nœuds en question (cf Tableau 28).

Notons bien que cost1 et cost2 sont les coûts affectés lorsqu'il s'agit de valeurs différents des opérateurs type1 (mandatory, optional) et type2 (or, alternative) respectivement. S'il sont de même valeur, on affecte 1.

	mandatory	optional	or	alternative	AND
mandatory	1	0.75	/	/	0.75
optional	0.75	1	/	/	0.75
or	/	/	1	0.5	0.5
alternative	/	/	0.5	1	0.5
AND	0.75	0.75	0.5	0.5	0

Ce tableau donne un exemple de distance entre les différents valeurs d'opérateurs. Il y a quartes valeurs : mandatory, optional, or, alternative. Par conséquent, en fonction de leur sémantique dans le contexte de service SaaS configurable, nous pouvons leur attribuer des coûts. Ces coûts peuvent être modifiés par l'utilisateur.

Algorithm : Simple Service Feature Model Matching (A,B)

1: if the roots of the two trees A and B contains distinct symbols then

2: Return 0 ;

3: else

4: m = the number of first level sub trees of A;

5: n = the number of first level sub trees of B;

6: Initialization : M[i,0] for i=0,..,m;

7: M[0,j] for j=0,..,m;

8: for i = 1 to m do

9: for j = 1 to n do

10: M[i, j] = max(M[i, j-1], M[i-1, j], M[i-1, j-1] + W[i, j]) where W[i, j] = SimpleTree-Matching (Ai, Bj) ;

11: end for

12: end for

13: if A and B are roots then

14: if Operator2 of A and B are distinct then

15: Matching Value =M[m][n]+1+cost2

16: else

17: Matching Value= M[m][n]+1+1

18: end if

19: end if

20: if ((A and B are leaves) or ((A is internal and B is leaf)or (A is leaf and B is internal)) then

21: if Operator1 of A and B are distincts then

22: Matching Value =M[m][n]+1+cost1

23: else

24: Matching Value= M[m][n]+1+1

25: end if

26: end if

27: if A and B are internal then

28: if Operator1 and Operator2 are distincts then

29: Matching Value =M[m][n]+1+cost1+cost2

30: else if Operator1 distinct and Operator2 have same value then

31: Matching Value =M[m][n]+1+cost1+1

32: else if Operator1 have same value and Operator2 distincts then

33: Matching Value =M[m][n]+1+1+cost2

34: else

35: Matching Value =M[m][n]+1+1+1

36: end if

37: end if

38: RETURN Matching Value

39: end if

Des modifications sur la formule de calcul de similarité s'ajoutent aussi et elle devient comme ceci :

$$\text{Similarité (A,B)} = \frac{SimpleServiceTreeMatching(A,B)}{(Size(A)+Size(B)+nbType1+nbType2)/2+nbType1'+nbType2'}$$

Où nbType1 c'est le nombre de nœuds ayant des opérateurs type1, nbType2 représente le nombre des nœuds ayant des opérateurs type2 dans les deux services à comparer, nbType1' et nbType2' pour nombre de nœuds qui portent l'operateur1 et l'operateur2 respectivement dans l'un des nœuds des deux services SaaS à comparer.

2.2. Algorithme Clustered Service Tree Matching

Le principe de cet algorithme est presque le même de SSTM sauf qu'il diffère au niveau d'attribution de la valeur de matching.

Algorithm Clustered Service Tree Matching (A,B)

1: if the roots of the two trees A and B contains distinct symbols then
2: Return 0 ;
3: else
4: m = the number of first level sub trees of A;
5: n = the number of first level sub trees of B;
6: Initialization : M[i,0] for i=0,..,m;
7: M[0,j] for j=0,..,m;
8: for i = 1 to m do
9: for j = 1 to n do
10: M[i, j] = max(M[i, j-1], M[i-1, j], M[i-1, j-1] + W[i, j]) where W[i, j] = SimpleTree-Matching (Ai, Bj) ;
11: end for
12: end for
13: if A and B are roots then
14: if Operator2 of A and B are distincts then
15: MatchingValue = [(M[m][n] * 1)/Max(Sibling(A); Sibling(B))] * cost2
16: else
17: MatchingValue = [(M[m][n] * 1)/Max(Sibling(A); Sibling(B))] *1
18: end if
19: end if
20: if ((A and B are leaves) or ((A is internal and B is leaf)or (A is leaf and B is internal)) then
21: if Operator1 of A and B are distincts then
22: MatchingValue = [(M[m][n] + 1)Max(Sibling(A); Sibling(B))] * cost1
23: else
24: MatchingValue = [(M[m][n] + 1)/Max(Sibling(A); Sibling(B))] * 1
25: end if
26: end if
27: if A and B are internal then

28: if Operator1 and Operator2 are distincts then

29: MatchingValue = [(M[m][n] * 1)/Max(Sibling(A); Sibling(B))] *cost1 * cost2

30: else if Operator1 distincts and Operator2 have same value then

31: MatchingValue = [(M[m][n] * 1)/Max(Sibling(A); Sibling(B))] *cost1 * 1

32: else if Operator1 have same value and Operator2 distincts then

33: MatchingValue = [(M[m][n] *1)/Max(Sibling(A); Sibling(B))] * 1 * cost2

34: else

35: MatchingValue = [(M[m][n] * 1)/Max(Sibling(A); Sibling(B))] * 1

36: end if

37: end if

38: RETURN Matching Value

39: end if

2.3. Algorithm Measuring Similarity service with Matrix

Le principe de fonctionnement de cet algorithme est de trouver des similitudes entre les arbres désordonnées efficacement. On a continué alors avec la structure d'un service SaaS configurable basée FM, et on a utilisé les même features (fonctionnalités) que les features dans le SSTM et le CSTM . Nous avons étendu le SWM d'origine en créant d'un nouveau arbre traité et résumés à partir l'arbre original et ses features pour ajouter des conditions et des priorités sur les nœuds de l'arbre. On extraire par la suite le niveau, le codage et le poids des informations de l'arbre résumé pour représenter les valeurs de cellule de la Matrice d'adjacence augmentée pour l'utiliser par la suite pour calculer la similarité.

Input: Two configurable SaaS trees CST1, CST2

Output: Measurement similarity between tree pair after the summarize

1. Create two new trees CST1'and CST2' summary of CST1 and CST2

2. Model the tree CST1' with the Augmented Adjacency

Matrix *A'*;

2. Model the tree CST2' with the Augmented Adjacency

Matrix *B'*;

3. **if** |*B'*|>|*A'*| **then**

Add (|*B'*| ─ |*A'*|) rows and columns of zeros

at the right end and bottom of the matrix *A'*;

else

Add (|*A'*| ─ |*B'*|) rows and columns of zeros

at the right end and bottom of the matrix *B'*;

end if

4. Calculate similarity between two configurable SaaS trees using

$$Cos(A',B') = \frac{\sum_{x=1}^{n}\sum_{y=1}^{n} A'_{xy} B'_{xy}}{\sqrt{\sum_{x=1}^{n}\sum_{y=1}^{n} A'^{2}_{xy}} \sqrt{\sum_{x=1}^{n}\sum_{y=1}^{n} B'^{2}_{xy}}}$$

3. Description de module de découverte

La requête générée par le client se compose de deux parties : partie abstraite et partie concrète. De même pour les SW exposées par les fournisseurs dans les registres. Les services des fournisseurs seront publiés dans des registres sur le Cloud.Pour que le client puisse trouver le service web qui répond à ses besoins, le programme de matching calcule la mesure de similarité entre son requête et tous les services présents dans le répertoire sur Cloud.

La valeur de similarité d'une requête d'un client avec un service de fournisseur s'obtient par la mesure de similarité de la partie abstraite et la partie concrète des deux services conformément à la formule suivante :

Similarity = α* AbstractSimilarity + β* ConcreteSimilary

Où α et β représentent les valeurs de pondération respectivement à la partie abstraite et la partie concrète. Dans ce travail, on a fixé $\alpha = 0.5$ et $\beta = 0.5$ Puis, selon la valeur calculée, le framework sélectionne et retourne les services web les plus similaires à la requête du client. Ceci est réalisable par la spécification d'un seuil pour la sélection. Pour que les services web des fournisseurs hébergés sur le Cloud servent plusieurs clients en parallèles, suivant l'architecture multi-tenant, on applique le principe de multi-threading. Dans ce cas, chaque tenant sera modélisée par un thread.

4. Etude de cas

Dans cette partie, on va présenter une étude de cas d'un service web fourni par une agence de voyage. Comme exemple illustratif de notre approche, considérons le cas d'un service SaaS configurable destiné à l'activité des agences de voyages moyennant des planificateurs des vacances afin d'organiser des voyages.

Les planificateurs des vacances sont conçus pour être utilisés par les applications Web mobiles pour la réservation d'hôtels, chambres, recherche de services supplémentaires tels que la location de voitures ou réservation d'activités (e.g., des attractions et des excursions.), convertir les frais de voyage avec la monnaie du pays de destination et payer pour les services choisis.

Le fournisseur de services offre un service SaaS configurable que les agences de voyages peuvent invoquer pour répondre aux besoins de leurs clients. Les agences de voyage sont considérés comme tenants tandis que leurs employés et clients joue le rôle d'utilisateurs appartenant à un locataire. Les clients de l'agence de Voyage peuvent accéder et vérifier l'état de l'offre de leurs Voyage en utilisant l'URL qui ramène au un nom de domaine personnalisé de l'agence de Voyage. Par défaut, il est possible de réserver des billets d'avion et de réaliser des paiements électroniques. De plus, le service offre l'option pour choisir la réservation d'hôtel, avoir un guide pour l'excursion aux lieux touristiques et historiques et un convertisseur de monnaies. En outre, le service exige que le paiement soit effectué par virement bancaire ou par carte de crédit. Il est possible d'effectuer une réservation de vol simple ou vol aller-retour. Une fois qu'un client a accomplit le processus de réservation, il reçoit une confirmation par courriel ou par message SMS dans les 24 heures. Les services SaaS configurables exposés par le fournisseur de services et le tenant sont représentés sur les figures 13, 14, 15 et 16.

Par exemple, la demande est initiée par un conférencier qui cherche pour un voyage intérieur pour assister à un congrès national. Il déclare la nécessite de transport de l'hôtel à l'aéroport comme option facultative alors que l'hôtel, la réservation vol aller-retour et le paiement en espèces comme exigences obligatoires.

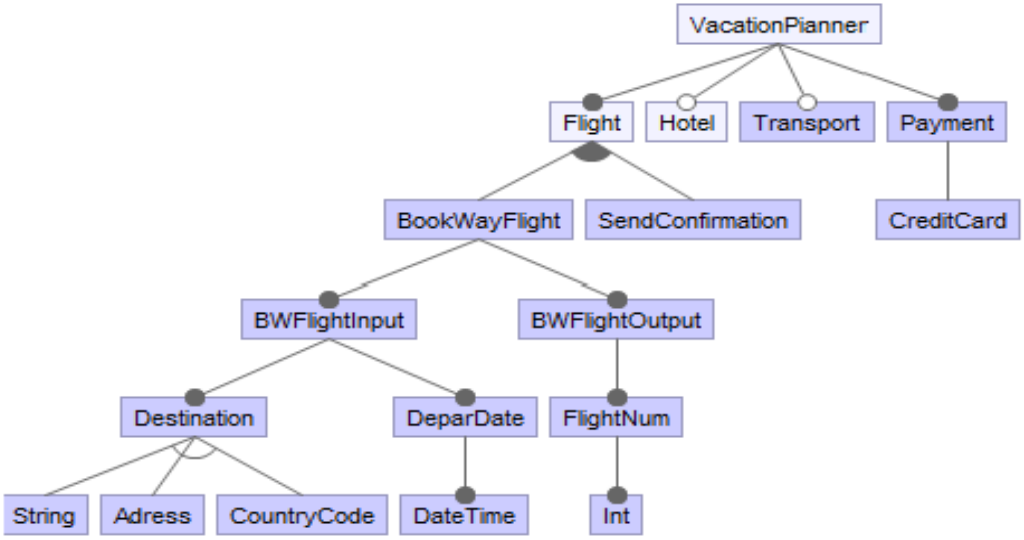

Figure 13:Partie abstraite de la requête

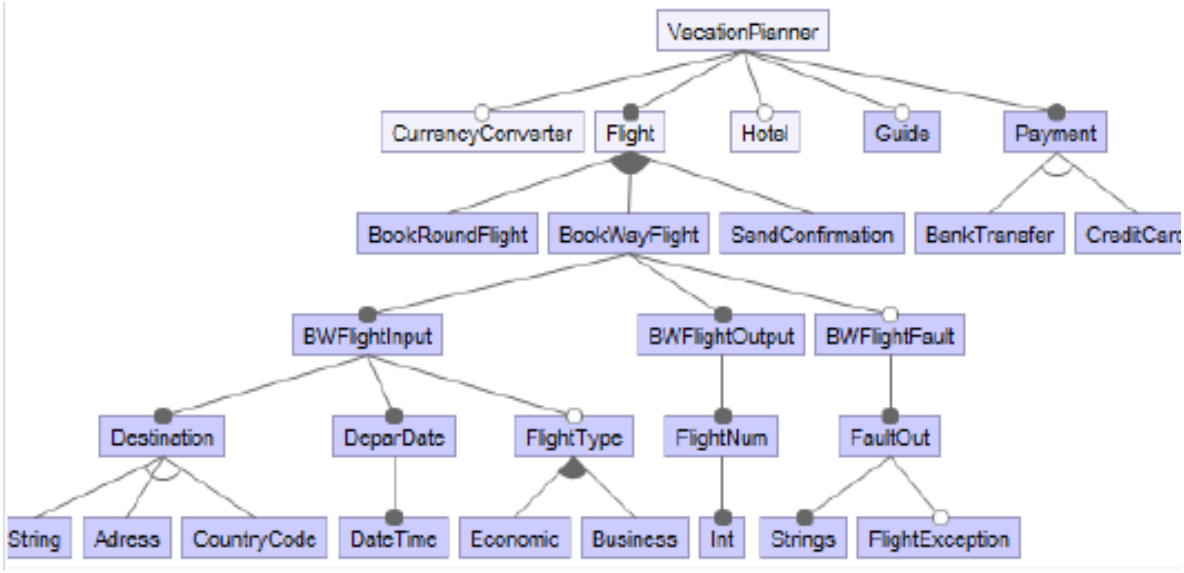

Figure 14:Partie abstraite de service web du fournisseur

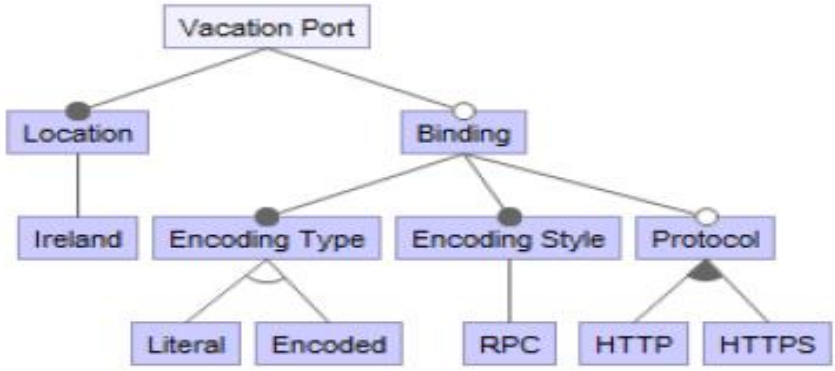

Figure 15:Partie concrète de la requête

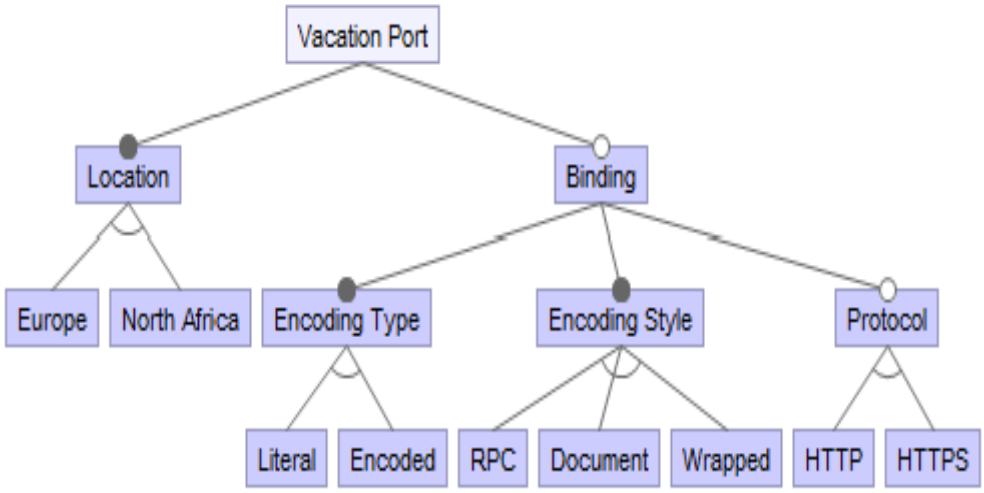

Figure 16:Partie concrète de service web du fournisseur

-Le tournage à la main du calcul de la similarité de la partie concrète est présenté ci-dessous :

Tableau 28: M 1, 13

	0	N14	N14-N15
0	0	0	0
N2	0	2.5	2.5
N2-N3	0	2.5	17.5

Tableau 29: W 1, 13

	N14	N15
N2	2.5	0
N3	0	15

Tableau 30: M 3, 5

	0	N18	N18-N19	N18-N20
0	0	0	0	0
N5	0	5	5	5
N5-N6	0	5	8.5	8.5
N6-N7	0	5	8.5	13

Tableau 31: W 3, 5

	N18	N19	N20
N5	5	0	0
N6	0	3.5	00
N7	0	0	4.5

Tableau 32: M 5, 18

	0	N21	N22
0	0	0	0
N8	0	1	1
N9	0	1	2

Tableau 33: W 5, 18

	N21	N22
N8	1	0
N9	0	1

Tableau 34: M 6, 19

	0	N23	N23-N24	N23-N25
0	0	0	0	0
N10	0	1	1	1

Tableau 35: W 6, 19

	N23	N24	N25
N10	1	0	0

Tableau 36: M 7, 20

	0	N26	N26-N27
0	0	0	0
N11	0	1	1
N11-N12	0	1	2

Tableau 37: W 7, 20

	N26	N27
N11	1	0
N12	0	1

Tournage à la main de CSTM :

Tableau 38: M 1, 13

	0	N14	N14-N15
0	0	0	0
N2	0	0	0
N2-N3	0	0	0.27

Tableau 39: W 1, 13

	N14	N15
N2	0	0
N3	0	0.27

Tableau 40: M 3, 5

	0	N18	N18-N19	N18-N20
0	0	0	0	0
N5	0	0.33	0.33	0.33
N5-N6	0	0.33	0.38	0.38
N6-N7	0	0.33	0.38	0.16

Tableau 41: W 3, 5

	N18	N19	N20
N5	0.3	0	0
N6	0	0.05	00
N7	0	0	0.16

Tableau 42: M 5, 18

	0	N21	N22
0	0	0	0
N8	0	0.5	0.5
N9	0	0.5	1

Tableau 43: W 5, 18

	N21	N22
N8	0.5	0
N9	0	0.5

Tableau 44: M 6, 19

	0	N23	N23-N24	N23-N25
0	0	0	0	0
N10	0	0.33	0.33	0.33

Tableau 45: W 6, 19

	N23	N24	N25
N10	0.33	0	0

Tableau 46: M 7, 20

	0	N26	N26-N27
0	0	0	0
N11	0	0.5	0.5
N11-N12	0	0.5	1

Tableau 47: W 7, 20

	N26	N27
N11	0.5	0
N12	0	0.5

Le tableau suivant illustre les résultats de calcul de similarité de l'exemple illustratif ainsi que le temps d'exécution pour les deux algorithmes :

Tableau 48: Tableau comparatif des résultats de calcul similarité

	Partie abstraite	Partie concrète	Total	Temps d'exécution
SSTM	0.81	0.82	0.81	7 ms
CSTM	0.35	0.27	0.31	6 ms
SSWM	0.43	0 .53	0.48	95 ms

5. Evaluation et experimentation

Dans cette partie, on s'intéresse à l'expérimentation et l'évaluation du modèle proposée ci dessus afin d'analyser son comportement et porter un jugement. L'évaluation quantitative, comme son nom l'indique, se base sur la quantité. Nous allons présenter les courbes d'évaluation du temps d'exécution par rapport au nombres de services web situées dans le registre en premier lieu, puis par rapport au nombre de tenants. A chaque fois on varie la quantité de services SaaS présents dans les registres et le nombre de tenants. Dans le premier test, on a fixé le nombre de tenants à 25 , puis on a augmentée à chaque fois le nombres de services SaaS dans le registre. La variation est faite par augmentation de 10 services à chaque fois jusqu'à atteindre 50 services. On a testée pour les trois algorithmes de matching SSTM , CSTM et SWM . Le résultat obtenu est représentée sous forme de courbes :

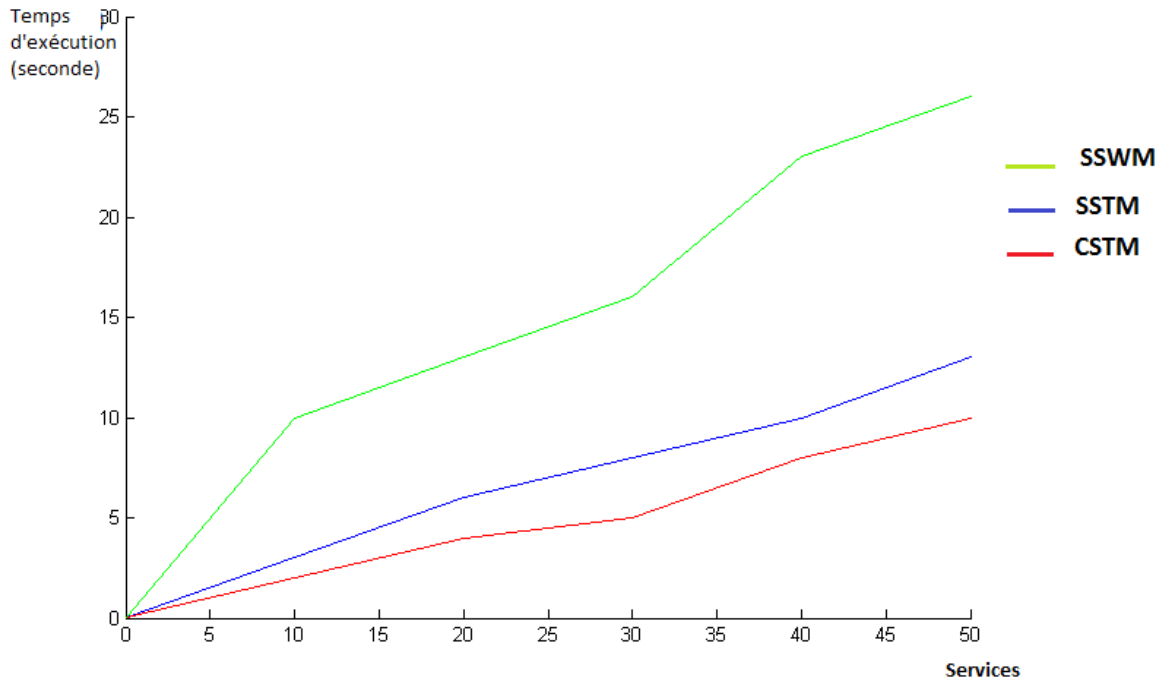

Figure 17:Evaluation temps de réponse pour 25 tenants fixe

Comme interprétation, on remarque que le temps de réponse du programme augmente d'une façon raisonnable à chaque fois qu'on augmente le nombre de services. Il atteint 13 secondes avec le SSTM , 10 secondes avec CSTM et 29 secondes avec SSWM. Donc dans ce cas, le SSTM est plus efficace que le CSTM et le SSWM.

Nous avons varie ensuite le nombre de tenants après avoir fixé le nombre de services héberges à 50 services. La variation se fait par l'augmentation de nombre de tenants par 5 jusqu'à atteindre 25 tenants. La courbe suivante montre le temps de réponse obtenu suite à l'augmentation des tenants.

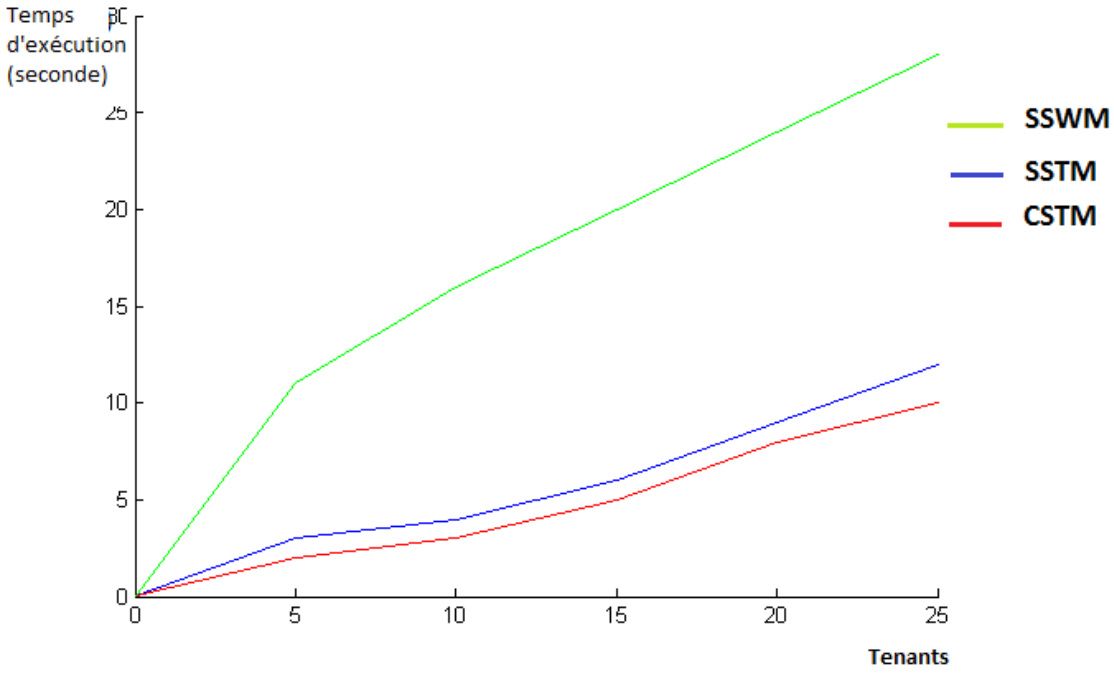

Figure 18:Evaluation temps de réponse pour 50 services fixe

On remarque que le temps de réponse atteint 28 secondes pour SWM, 12 secondes pour le SSTM et 10 secondes pour CSTM. Donc lorsqu'on varie le nombre de tenants, le CSTM répond plus rapidement par rapport au SSTM et SSWM.

Conclusion et perspectives

La gestion de la variabilité, le formalisme FODA FM et la sensibilité au multi-tenancy sont des solutions clés pour la réalisation des économies d'échelle à travers la réutilisation et la découverte de services SaaS configurables.En raison de leur nature variabilité intrinsèque et de la complexité engendrée par le processus de personnalisation, il y a une rupture entre l'efficacité de la découverte de services et les exigences personnalisables des tenants.

Dans ce Travail, une approche de découverte de services SaaS configurables est initiée pour renforcer la réutilisation dans un contexte Cloud multi-tenant.Pour les travaux futurs, nous comptons inclure les attributs du contrat SLA (Service Level Agreement) et les capacités de négociation à base d'argumentations [9] dans un Framework de publication/découverte de services configurables. Ce Framework peut être ensuite enrichi avec des outils pour permettre à un fournisseur de caractériser facilement et exposer ses services offerts et à un tenant de décrire facilement ses exigences en termes de services à retrouver. L'approche à proposer est susceptible d'être généralisée pour tenir compte un sous ensemble de tenants en négociation avec un sous ensemble de fournisseurs.

Bibliographie

[1] https en.wikipedia.org wiki cloud computing.

[2] https fr.wikipedia.org wiki architecture orienté services.

[3] Douglas K Barry. Web services, service-oriented architectures, and cloud computing: The savvy manager's guide (the savvy manager's guides), 2012.

[4] Boutaina Chakir and Mounia Fredj. Sv3r : un Framework pour la gestion de la variabilité des services. Revu, page 20, 2014.

[5] Cor-Paul Bezemer and Andy Zaidman. Multi-tenant saas applications : maintenance dream or nightmare ? In Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE).

[6] Peter Mell and Tim Grance. The nist definition of cloud computing. 2011.

[7] Clément Quinton, Daniel Romero, and Laurence Duchien. Saloon : a platform for selecting and configuring cloud environments.

[8] Kuo-Chung Tai. The tree-to-tree correction problem

[9] Fabricia Roos-Frantz. A preliminary comparison of formal properties on orthogonal variability model and feature models. VaMoS, 29 :121{126, 2009.}

[10] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson.Feature-oriented domain analysis (foda) feasibility study. Technical report, DTIC Document,1990.