Semantic based cloud broker architecture optimizing users satisfaction

Inès Fakhfakh

▶ **To cite this version:**

Inès Fakhfakh. Semantic based cloud broker architecture optimizing users satisfaction. Other [cs.OH]. Institut National des Télécommunications, 2015. English. NNT : 2015TELE0008 . tel-01166538v2

HAL Id: tel-01166538

https://theses.hal.science/tel-01166538v2

Submitted on 23 Jun 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THESE DE DOCTORAT CONJOINT TELECOM SUDPARIS et L'UNIVERSITE PIERRE ET MARIE CURIE

Spécialité : Informatique et Réseaux

Ecole doctorale : Informatique, Télécommunications et Electronique de Paris

Présentée par

Inès FAKHFAKH

Pour obtenir le grade de
DOCTEUR DE TELECOM SUDPARIS

UNE ARCHITECTURE DE CLOUD BROKER BASEE SUR LA SEMANTIQUE POUR L'OPTIMISATION DE LA SATISFACTION DES UTILISATEURS

Soutenue le 07/05/2015

devant le jury composé de :

Mme. Lynda MOKDAD, Professeur à la Faculté des Sciences et Technologie, Université Paris 12, Val de Marne – Rapporteur
M. Yacine GHAMRI DOUDANE, Professeur à l'Université de La Rochelle – Rapporteur
Mme Véronique VEQUE, Professeur à l'Université Paris-Sud – Examinateur
M. Pierre SENS, Professeur à l'Université Paris 6 – Examinateur
Mme. Amel BOUZEGHOUB, Professeur à Télécom SudParis – Examinateur
M. Djamal ZEGHLACHE, Professeur à Télécom SudParis – Directeur de thèse

Thèse n° 2015TELE0008

I dedicate this thesis to my family for their constant support and unconditional love.

Acknowledgements

I am very grateful to my reading committee members: Prof. Lynda MOKDAD and Prof.Yacine GHAMRI DOUDANE for accepting to judge this work. Thank you for your time, your interest, and your helpful comments. I would also like to thank the two other members of my oral defense committee Prof. Véronique VEQUE and Prof. Pierre SENS for their time and insightful questions.

I owe my deepest gratitude to Prof. Amel BOUZEGHOUB for all her contributions of time, ideas, and guidance, especially in the semantic part of this thesis.

My special thanks go to Prof. Djamal ZEGHLACHE for offering me the opportunity to work on this thesis at Telecom Suparis Institute and for all the advice he provided to the work on this thesis.

I would like to thank my colleagues and friends, not only for being supportive during the work on this thesis but also for the fun times we spent together.

Last but not the least, I would like to thank my family for supporting me throughout my life.

Abstract

Cloud computing is a dynamic new technology that has huge potentials in enterprises and markets. Its resources configuration changes significantly depending on various internal modifications such as fail of some components or external environment changes. The dynamicity and the increasing complexity of cloud architectures involve several management challenges. In this work, we are interested in the service level agreement (SLA) management. Actually, there is no standard to express cloud SLA, so, providers describe their SLAs in different manner and different languages, which leaves the user puzzled about the choice of its cloud provider.

To overcome these problems, we introduce a cloud broker architecture managing the service level agreements between providers and consumers. It aims to assist users in establishing and negotiating SLA contracts and to help them in finding the best provider that satisfies their service level expectations. Our broker SLA contracts are formalized as OWL ontologies as they allow to hide the heterogeneity in the distributed cloud environment and enable interoperability between cloud actors. Besides, by combining our ontology with our proposed inference rules, we contribute to detect violations in the SLA contract assuring thereby the sustainability of the user satisfaction. Based on the requirements specified in the SLA contract, our cloud broker assists users in selecting the right provider using a multi attribute utility theory method. This method is based on utility functions representing the user satisfaction degree. To obtain accurate results, we have modelled both functional and non functional attributes utilities. We have used personalized utilities for each criterion under negotiation so that our cloud broker satisfies the best consumer requirements from functional and non functional point of view.

Résumé

Le cloud computing est un nouveau modèle économique hébergeant les applications de la technologie de l'information. Il répond aux besoins exponentiellement croissants en ressources physiques et logicielles. Il permet d'approvisionner ces ressources et de les partager sous forme de ressources virtuelles à travers le réseau internet. Le passage au cloud devient un enjeux important des entreprises pour des raisons essentiellement économiques. En effet, le cloud fournit des services à la demande ce qui permet aux utilisateurs d'allouer les ressources virtuelles nécessaire pour leurs processus métier. Ils n'ont plus donc besoin d'installer des infrastructures couteuses et d'assurer leurs mis à jours ce qui permet de réduire les coûts de l'exploitation et de l'entretien. En outre, le cloud assure la flexibilité, la scalabilité, la fiabilité et la haute disponibilité.

Aujourd'hui il existe plusieurs fournisseurs de service cloud public, privé et hybride.

Étant donné la diversité des fournisseurs et des services du cloud, l'utilisateur doit être capable de sélectionner celui qui répond au mieux à ses besoins. Ce choix n'est pas évident car l'utilisateur doit être en mesure de trouver le meilleurs compromis entre plusieurs paramètres de qualité de service proposés par les différents fournisseurs de services. Il doit aussi prendre en compte la dynamicité de l'infrastructure du cloud. En outre, l'hétérogénéité syntaxique et sémantique entre les fournisseurs du cloud rends ce choix encore plus difficile. Cette hétérogénéité se manifeste en :

- différentes terminologies exprimant le même concept : par exemple, la capacité de traitement d'une machine virtuelle peut être appelé CPU, capacité, capacité de traitement ... ;

- Différents formats ou différents langages exprimant la requête : certain fournisseurs proposent des instances prédéfinies et c'est à l'utilisateur de composer sa requête à partir de ces instances alors que d'autres exigent que l'utilisateur fixe les caractéristiques techniques des ressources à allouer tel que sa capacité, sa mémoire son espace disque ...
- Différentes façons d'exprimer la qualité de service : dans certains cas deux métriques différentes peuvent avoir la même appellation. Par exemple, le mot "delay" peut faire référence à la latence d'un système comme il peut exprimer le pourcentage des paquets perdu lors de la transmission d'un service.

Afin de définir l'accord et la nomenclature utilisée entre le fournisseur et le consommateur de service, on crée des contrats SLA (Service Level Agreements). Le contrat SLA est un contrat formel entre les fournisseurs et les consommateurs du cloud assurant la qualité de services négociée. Les contrats SLA actuellement établis sont basés sur les modèles proposées par les fournisseurs de service. A cause de la diversité des terminologies et des langages, la création et la négociation de contrat SLA avec plusieurs fournisseurs devient une tâche assez difficile pour l'utilisateur. Ce dernier doit s'adapter à chaque fois avec les terminologies et les langages de description du SLA de chaque fournisseur ce qui rends la comparaisons des différents offres une tache assez compliquée. Par conséquent, l'utilisateur peut facilement se tromper dans son choix final du meilleur fournisseur. En outre, une fois le contrat SLA établit, les fournisseurs du cloud ne proposent aucune garantie de performance. En fait, c'est à l'utilisateur de détecter les violations dans le contrat SLA et de s'assurer que les pénalités imposées dans le contrat en cas de violation soient exécutées.

L'objectif de cette thèse est de :

1. Assurer l'interopérabilité : prendre en compte la sémantique de la qualité de service dans le processus de négociation de contrat SLA ;

2. Garantir une satisfaction optimale : trouver le meilleur compromis qui répond au besoin de l'utilisateur ;
3. Garantir le respect de la satisfaction client dans le temps : fournir à l'utilisateur des moyens lui permettant de surveiller le respect de ses exigences.

Afin d'atteindre ces objectifs, nous allons :

1. Définir une ontologie spécifique au cloud décrivant le contrat SLA
2. Proposer une méthode multi-critère qui permet de sélectionner le fournisseur qui répond au mieux à la requête de l'utilisateur
3. Proposer une politique qui permet de détecter les violations dans le contrat SLA en s'appuyant sur des règles et de l'inférence

Ces différentes contributions sont structurés dans une architecture globale appelée "Cloud Broker Architecture". L'objectif de cette architecture est d'aider les utilisateurs à trouver des services cloud répondant au mieux à leur requête. Pour ceci, le broker utilise une méthode multicritère qui permet de trouver le meilleur offre en prenant en compte plusieurs critères de choix. Dans cette architecture, le contrat SLA est exprimé à l'aide d'une ontologie assurant l'interopérabilité entre les différents acteurs du cloud. Ce contrat est géré à l'aide de politiques, exprimé sous forme de règles permettant de détecter les violations du contrat.

La structure de cette thèse est la suivante.

Le premier chapitre présente l'état de l'art en introduisant le concept de cloud computing, ses caractéristiques, ses modèles de service, ses modèles de déploiement et de ses principaux défis de gestion. Il introduit aussi quelques outils de gestion sémantiques comme l'ontologie et les règles et présente des travaux qui ont utilisés les ontologies pour la découverte de services. Dans ce chapitre, nous présentons aussi des travaux qui ont proposé des méthodologies pour la sélection de services. Finalement, nous définissons la notion de "cloud broker" et nous présentons quelques travaux qui ont proposé des architectures de broker. Dans le chapitre 2, nous introduisons l'architecture

de broker que nous avons proposé, ses composantes et ses principales caractéristiques. Nous présentons également l'ontologie utilisée pour définir le contrat SLA et nous expliquons les avantages de l'utilisation des règles de gestion. Dans ce travail, nous avons utilisé une méthode multi-critère pour sélectionner le meilleur fournisseur de service que nous expliquons dans le chapitre 3. Ce chapitre introduit aussi les fonctions d'utilités que nous avons utilisé pour calculer le degré de satisfaction des utilisateurs. Finalement, le chapitre 4 explique l'avantage de l'utilisation des annotations sémantique dans la découverte de services et nous évaluons les fonctions d'utilités que nous avons proposé.

Dans la suite de ce résumé, nous présentons le travail réalisé dans les trois chapitres suivants le chapitre état de l'art :

Une Architecture de Broker pour la Négociation de Contrats SLA
Afin de remédier aux problématiques mentionnées ci dessus, nous avons proposé une architecture de Broker cloud. Ce broker joue le rôle d'intermédiaire entre les fournisseurs et les consommateurs du cloud. C'est une interface de gestion unique permettant de traiter les différentes requêtes des utilisateurs, établir des contrats SLA entre les clients et les fournisseurs de service et d'assurer le respect des contrats établit. Cette architecture est illustrée par la figure 1.

Les différents modules de cette architecture sont :

- Broker Manager : c'est le module principale du broker qui assure l'orchestration des différentes opérations. Il englobe un module "Request Parser" qui fournit au utilisateurs du broker un modèle de requête à remplir puis parse la requête de l'utilisateur en ontologie. Il est aussi muni d'un module "QoS Parser" qui interagit avec les fournisseurs de services en leurs envoyant la requête de l'utilisateur et en recevant les QoS qu'ils peuvent fournir.
- SLA Contract Ontology : décrit la structure du contrat SLA établit entre les utilisateurs et les fournisseurs du cloud.

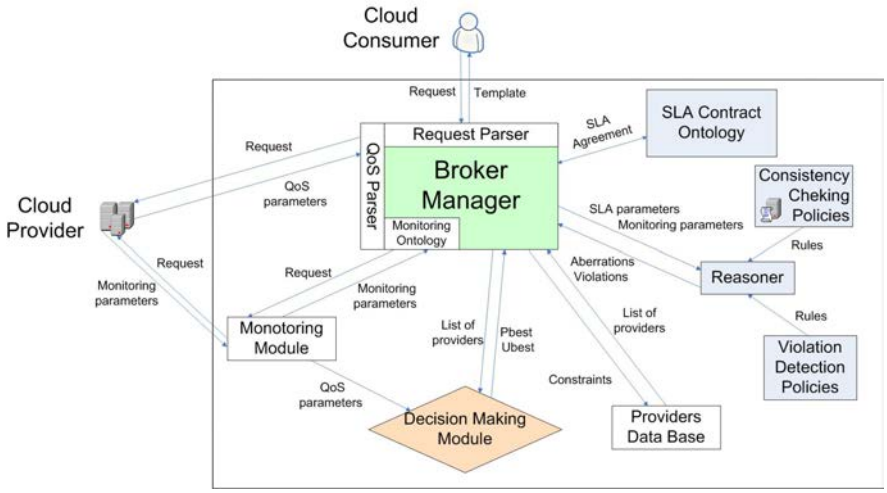

Figure 1: Architecture de Cloud Broker -

- Reasoner module : ce module assure la vérification de la cohérence de la requête de l'utilisateur et la détection des violations dans le contrat SLA.
- Consistency Checking Policies module : contient des règles utilisées pour vérifier la cohérence des requêtes.
- Violation Detection Policies module : contient des règles utilisées pour détecter les violations dans le contrat SLA et pour déterminer les pénalités à imposer au fournisseur de service en cas de violation.
- Decision Making Module : ce module est chargé de trouver le fournisseur qui propose le meilleur offre pour la requête de l'utilisateur. Il évalue les performances de chaque fournisseur en utilisant une méthode multi-critère.
- Providers knowledge Base : il s'agit d'une base de donnée énumérant les différents fournisseurs du cloud. Elle contient des informations sur ces fournisseurs comme leur méthode de réplication, leur stratégie de récupération et la localisation géographique de leurs données stockés.
- Monitoring module : afin de détecter les violations dans le contrat SLA, nous avons prévu un module de monitoring qui permet de mesurer les valeurs de la qualité de service en temps réel. Si ces valeurs n'honore

pas l'accord établit entre le consommateur et le fournisseur de service, des pénalités sont imposés au fournisseur.

Cette architecture assure trois fonctions :

1. Établir un contrat SLA cohérent : les utilisateurs du broker ne sont pas forcément des personnes expérimentées dans le domaine du cloud capables de formuler des demandes cohérentes. Ainsi, nous avons prévu de vérifier la cohérence sémantique de la requête de l'utilisateur avant d'entamer le processus de sélection du meilleur offre pour cette requête. Pour ce faire, le module "broker manager" parse la requête sous forme d'ontologie puis demande au module "reasoner" de vérifier sa consistance. Ce module vérifie si les règles exprimés dans le module "Consistency Checking Policies" sont respectées par la requête. Dans le cas contraire, le broker alerte l'utilisateur pour qu'il reformule sa demande. Les règles stockées dans le module "Consistency Checking Policies" sont prédéfinit par le gestionnaire du broker en adéquation avec les concepts de l'ontologie utilisée pour exprimer le contrat SLA.
2. Sélectionnez le meilleur fournisseur pour la requête de l'utilisateur et établir le contrat SLA : Le principal objectif du broker est de trouver le meilleur offre du fournisseur de service qui correspond à la demande de l'utilisateur et établir un contrat SLA entre le consommateur et le fournisseur sélectionné. Ce processus passe par plusieurs étapes :
    - le broker sélectionne les fournisseur de service susceptible de répondre à la requête de l'utilisateur en éliminant les fournisseurs qui ne peuvent pas répondre aux contraintes dures de l'utilisateur comme le type de réplication demandé et la localisation géographique des données stockées.
    - en se basant sur les performances des fournisseurs de services constatées en exécutant d'anciennes requêtes, le broker prédit les performances de chaque fournisseur de service sélectionné pour la nouvelle requête. Ensuite, il compare la qualité de service fourni par les différents fournisseurs et sélectionne celui qui répond au mieux à la requête de l'utilisateur.

- le broker contacte le fournisseur de service sélectionné et établit le contrat SLA entre avec le consommateur.

3. Détecter les violations dans le contrat SLA : après avoir établit le contrat SLA entre le consommateur et le fournisseur de service, le broker assure que les termes de ce contrat soient respectées. En effet, il vérifie régulièrement que la qualité de service assurée par le fournisseur en temps réel satisfait le consommateur. Pour ce faire, il fait encore une fois appel au raisonneur qui compare les valeur des paramètre QoS au seuils fixés par le contrat SLA. En cas de violation, il s'assure que des pénalités soient imposées au fournisseur de service. Les pénalités sont déterminées au préalable par le gestionnaire du broker sous forme de règles et stockées dans le module "Violation Detection Policies".

Afin d'assurer ces différentes fonctionnalités, nous décrivons sémantiquement le contrat SLA entre l'utilisateur et le fournisseur du cloud sous forme d'ontologie. Nous proposons une ontologie générique pour décrire un contrat SLA dans le domaine du cloud. Ce contrat est établit entre un utilisateur qui doit spécifier sa requête, ses contraintes et les critères de qualité de service dont il a besoin, et un fournisseur de service qui spécifie les valeurs de QoS qu'il s'engage à fournir pour cette requête. La durée de ce contrat est déterminée par l'attribut "Duration". La structure de l'ontologie représentant le contrat SLA est illustrée par la figure 2.

Cette ontologie générique est spécifiée pour la couche infrastructure du cloud. En effet, la requête de l'utilisateur de la couche infrastructure doit être formulé sous forme de demande de ressources virtuelles. La formulation de la requête que nous avons proposé est conforme au standard OCCI ce qui permet d'assurer l'interopérabilité entre les différents acteurs du cloud et facilite de faire la correspondance entre notre ontologie et les autres formalismes utilisés par les fournisseurs du cloud. De plus, nous définissons dans cette ontologie les contraintes et les critères de la qualité de service qui intéressent un utilisateur de la couche infrastructure du cloud comme illustré par le figure 3.

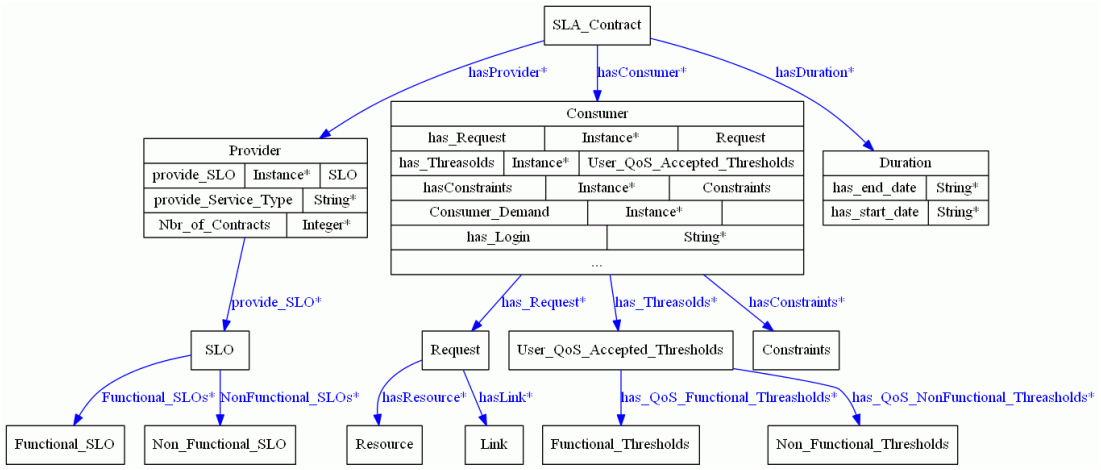

Figure 2: Notre Ontology représentant le contrat SLA -

L'utilisateur de la couche IaaS du cloud peut donc s'adresser à notre broker afin de chercher un fournisseur de services cloud en utilisant le modèle proposée par notre ontologie. Ainsi, il évite de confronter les différents fournisseurs du cloud qui utilisent des terminologies différentes. Toutefois, le broker doit communiquer avec plusieurs fournisseur de service cloud afin de négocier avec eux les contrats SLA. Ces fournisseurs utilisent différents langages de description du contrat SLA. Afin de permettre au broker de communiquer avec les fournisseurs de service, nous proposons de faire un "mapping" entre les langages de description du contrat SLA utilisé par ces fournisseur et notre ontologie. Vu que le langage WS-Agreement est un standard largement utilisé, nous proposons une méthode permettant de faire la correspondance entre notre ontologie et ce langage. L'objectif de la spécification du WS-Agreement est de définir un langage générique permettant de créer tout type d'accord quelque soit le domaine d'application, et il a été utilisé dans le domaine du cloud. La structure du WS-Agreement est illustrée par la figure 4. Elle est composée de :

- son nom
- le contexte décrivant les deux parties de l'accord et sa date d'expiration
- les termes de ce contrat comprenant la description du service et les garanties que le fournisseur s'engage à assurer pour ce service. Le service

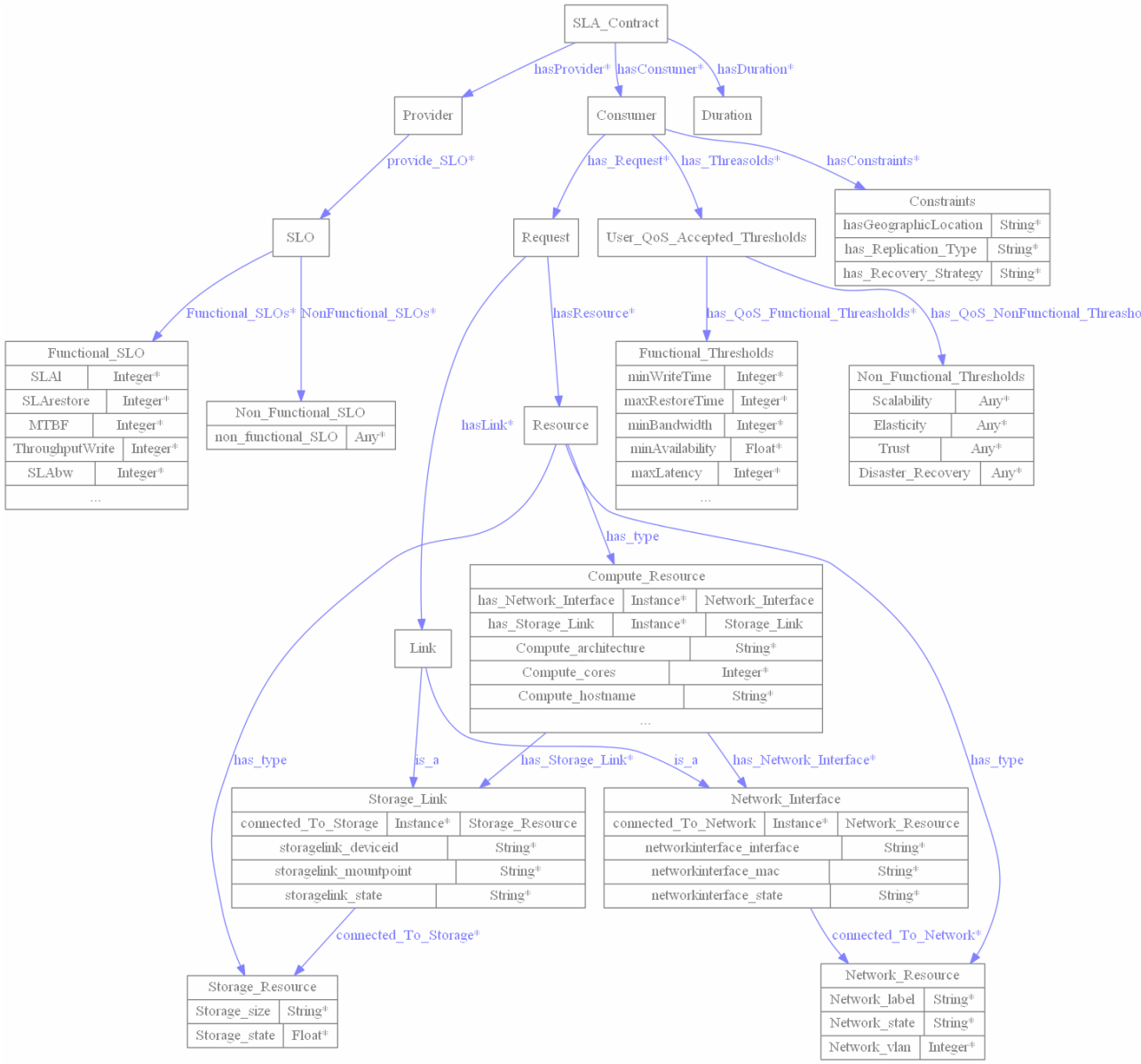

Figure 3: Notre Ontologie représentant le contrat SLA pour la couche infrastructure du cloud -

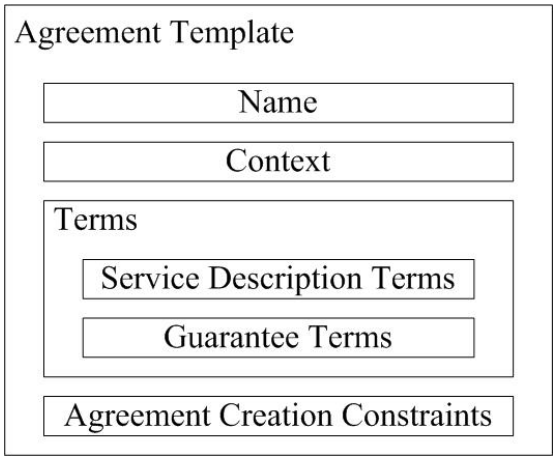

Figure 4: Structure du langage WS-Agreement -

est définit par un ensemble de propriétés qui correspondent au critères à négocier. Les garanties intègrent à la fois les indicateurs de performances que les fournisseur devra assurer et les pénalités à imposer au fournisseur en cas de violation. Dans le cadre de notre ontologie, le service négocié corresponds à la requête à satisfaire, et les propriétés de cette requêtes sont les paramètres "SLO " de notre ontologie. Les valeurs numériques de ces paramètres représentent les performances que le fournisseur est en mesure de garantir, et les pénalités sont introduite dans les règles d'inférence qui permettent de détecter les violations.

- les contraintes à respecter lors de l'établissement du contrat SLA qui décrivent les valeurs acceptables pour chaque critère négocié. Ces valeurs sont directement intégré dans l'ontologie lors de la définition de chaque paramètre.

Nous montrons par ce "mapping" que l'ontologie proposée associé au règles d'inférence contient tous les éléments d'un WS-Agreement. L'ontologie introduit le contexte, la description de service et les critères négocié comme illustré par la figure 5. Les règles d'inférence définissent les pénalités à imposer en cas de violation. Notre système est encore plus riche qu'un WS-Agreement puisque il garde la trace des préférences des utilisateurs et introduit une méthode qui permet de détecter les violations du contrat.

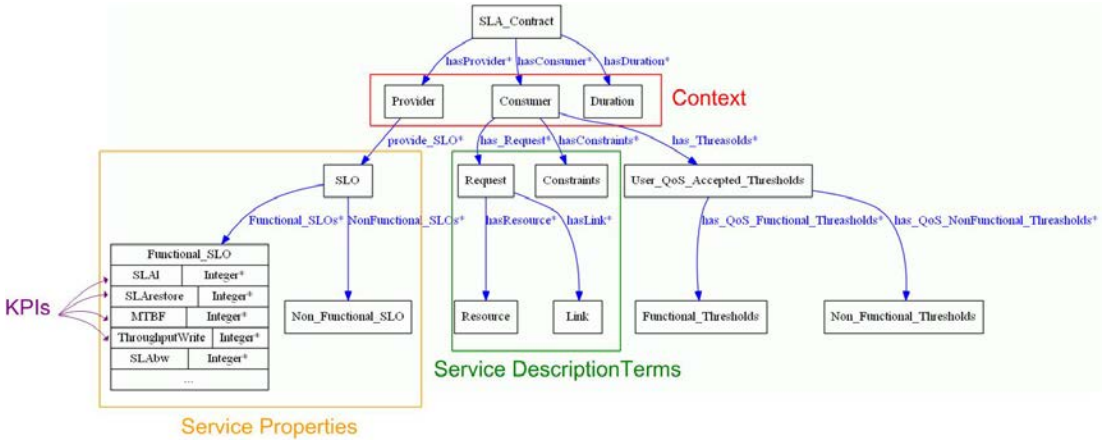

Figure 5: Le mapping entre notre ontologie et le language WS-Agreement -

La Sélection du meilleur fournisseur de service cloud en se basant sur une méthode multi-critère Le cloud computing est un nouveau paradigme qui a attiré plusieurs clients cherchant à profiter de la haute disponibilité des ressources du cloud et de minimiser le coût d'exploitation des services. Ces clients ont différents secteurs d'activité, par conséquent ils ont des besoins et des attentes différentes des service offert par les fournisseurs du cloud. Ils ont donc besoin de comparer soigneusement les services offert par les différents fournisseurs et de trouver le meilleur compromis qui réponds à leur demande. Dans ce travail, nous avons proposé une architecture de broker qui a pour objectif d'aider les utilisateur à choisir leur fournisseur. Il utilise une méthode multi-critère qui choisit le meilleur fournisseur en se basant sur les critères définit par l'utilisateur et sur ces préférences. Cette méthode est appelé MAUT (Multi-Attribute Utility Theory). Cette méthode considère chaque critère comme une fonction d'utilité à intégrer dans une super-fonction d'utilité. Il existe trois formes différentes pour appliquer la méthode MAUT : multi-linéaire, additive et multiplicative. Selon Keeney et Raiffa (1), les formes additive et multiplicatives sont plus appropriés que la forme mutli-linéaire si on considère plus que quatre critères. De plus, la forme additive est adoptée lorsque l'interaction entre les attributs n'est pas importante. Dans le contexte du cloud, plusieurs paramètres présentant une forte interaction sont mis en jeu dans le choix du meilleur service. En se ba-

sant sur la forme multiplicative de la méthode MAUT, nous avons proposé un algorithme qui permet de sélectionner le meilleur service proposé par un fournisseur cloud. Cet algorithme est présenté ci dessous :

La requête de l'utilisateur est représentée par le vecteur R :

$$\begin{aligned} &R = [r_1, r_2, \cdots, r_n] \\ &n : \ nombre\ de\ parametres\ dans\ la\ requete \end{aligned} \tag{1}$$

L'ensemble des fournisseurs de services capables de répondre à cette requête est représenté par le vecteur P :

$$\begin{aligned} &P = [p_1, p_2, \cdots, p_m] \\ &m : \ nombre\ de\ fournisseurs\ de\ services \end{aligned} \tag{2}$$

Les critères négocié sont représentés par le vecteur C :

$$\begin{aligned} &C = [c_1, c_2, \cdots, c_k] \\ &k : \ nombrede\ criteres \end{aligned} \tag{3}$$

Le degré d'importance de chaque critère pour l'utilisateur est spécifié sous forme de poids. Les poids relatives aux critères négociés sont représentés par un vecteur W.

$$W = [w_1, w_2, \cdots, w_k] \tag{4}$$

L'utilité de chaque fournisseur est calculée par l'équation suivante :

$$\begin{aligned} &\forall i \in [1, m], U(p_i) = \frac{1}{w}[\prod_{j=1}^{k}(1 + ww_j f_j(c_j)) - 1)] \\ &where\ (1 + w) = \prod_{j=1}^{k}(1 + ww_j) \end{aligned} \tag{5}$$

l'utilité globale de chaque fournisseur est calculée à partir des utilités unitaire des différents critères de la qualité de service et des poids attribués

par l'utilisateur à ces critères. Ainsi, notre broker peut sélectionner le fournisseur qui propose le meilleur service en se basant sur les préférences des utilisateurs. Ce service est celui qui obtient la meilleur utilité globale $U(p_i)$.

Le choix du meilleur fournisseur de service est basé essentiellement sur les fonctions d'utilité permettant de représenter le degré de satisfaction client vis à vis à chaque critère de la qualité de service. Plusieurs travaux ont adopté cette méthode pour résoudre des problématique de choix multi-critère. Néanmoins, ils utilisent la même forme de courbe (courbes exponentielles, logarithmiques, linéaire, hyperbolique ...) pour représenter tout les critères de choix. Dans ce travail, nous proposons d'illustrer chaque fonction d'utilité par une courbe spécifique à lui représentant bien le critère de choix. Nous essayons de cerner la plupart des critères qui contribuent dans le choix du meilleurs service et de les classer en critères fonctionnels et critères non fonctionnels. Dans ce chapitre, nous modélisons les fonctions d'utilités de ces différents critères.

Expérimentations et résultats Dans ce travail, nous avons proposé une architecture de broker aidant les utilisateurs à trouver le meilleur fournisseur satisfaisant leur demande, en fonction de leur préférences. Afin d'atteindre cet objectif, le broker assure une meilleure compréhension et une interopérabilité entre les différents acteurs du cloud en adoptant les annotations sémantiques. En fait, il utilise l'ontologie et les règles d'inférences pour définir le contrat SLA entre les consommateurs et les fournisseurs de services cloud et pour détecter les éventuelles violations de ce contrat. Afin d'aider les utilisateurs à choisir le meilleur fournisseur de service avec lequel ils peuvent établir des contrats SLA, le broker utilise une méthode multi-critère appelée la méthode MAUT qui est basée sur les fonctions d'utilités. Nous avons modélisé les fonctions d'utilité des critère fonctionnels et non fonctionnels négocié dans le contrat SLA. Ces fonctions permettent d'évaluer le degré de satisfaction de l'utilisateur par rapport à chaque critère négocié.

Afin d'évaluer notre solution, nous avons implémenté notre architecture de boker en langage java. Nous avons utilisé le logiciel Protégé et la bibliothèque

Jena pour créer notre ontologie représentant le contrat SLA et pour définir les règles d'inférence. Ensuite, nous avons réalisé trois expériences :

Évaluation des annotations sémantiques Afin d'évaluer l'avantage des annotations sémantiques, nous avons réalisée l'expérience illustrée par la figure 6. Cette expérience consiste à trouver le service qui répond à la requête de l'utilisateur parmis trois services proposés par trois fournisseurs utilisant des terminologies différentes. De plus chaque fournisseur utilise une unité différente pour mesurer la capacité du processeur. Par exemple, le premier fournisseur adopte les instances Amazon ECU (1 ECU = 1.0-1.2 GHz) alors que le deuxième fournisseur utilise des processeurs Intel Xeon E5520 qui ont une capacité équivalente à 4.52 GHz.

Figure 6: Le scénario de l'évaluation sémantique -

Dans un premier temps, nous avons effectué une recherche syntaxique qui compare les critères de la requête de l'utilisateur à ceux proposés par les fournisseurs de services. Les résultats sont illustrés par la figure 7. Nous

remarquons que seuls le "CPU" et le "Delay" ont été découvert car on retrouve ces mêmes termes dans la requête et les propositions des fournisseurs. De plus l'utilité attribué à la capacité du processeur n'est pas fiable car elle ne tient pas en compte les différences entre les unités. On remarque aussi que le terme "Delay" n'a pas la même signification pour l'utilisateur et le fournisseur. Ce qui fait qu'on obtient le résultat suivant : $Utility(P3) > Utility(P1) > Utility(P2)$. Dans un deuxième temps, nous avons introduit les annotations sémantiques qui permettent de découvrir tout les critères et de masquer les hétérogénéité syntaxiques et sémantiques. Dans ce cas, nous obtenons le résultat suivant : $Utility(P2) > Utility(P1) > Utility(P3)$ (figure 8) ce qui prouve que l'absence d'annotations sémantique peut conduire à choisir un fournisseur qui ne peut pas satisfaire la requête de l'utilisateur.

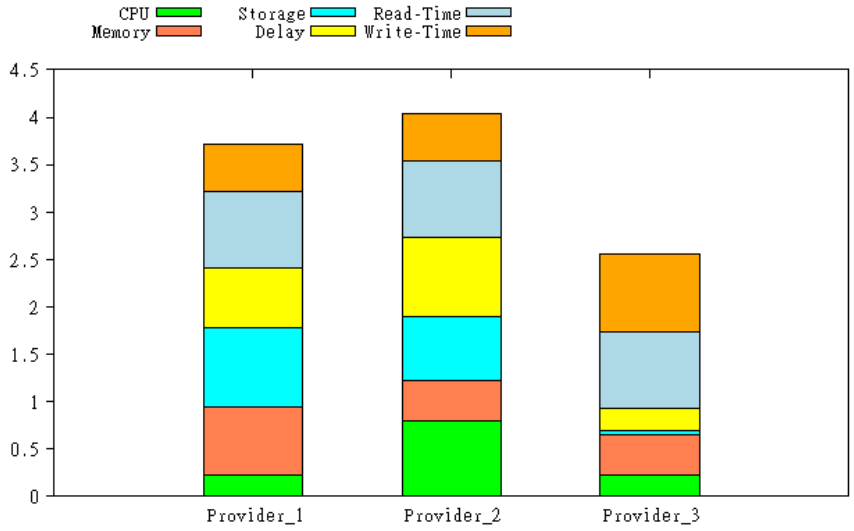

Figure 7: Les résultats de la recherche syntaxique -

Évaluation des fonctions d'utilité fonctionnelles Les fonctions d'utilité peuvent avoir plusieurs formes : exponentielle, logarithmique, linéaire, hyperbolique etc. Dans ce travail, nous avons choisi de modéliser chaque critère par une fonction d'utilité propre à lui. Afin d'évaluer notre méthode, nous avons pris l'exemple d'une requête composée de trois critères : temps de réponse, latence et bande passante et de dix offres de fournisseur de ser-

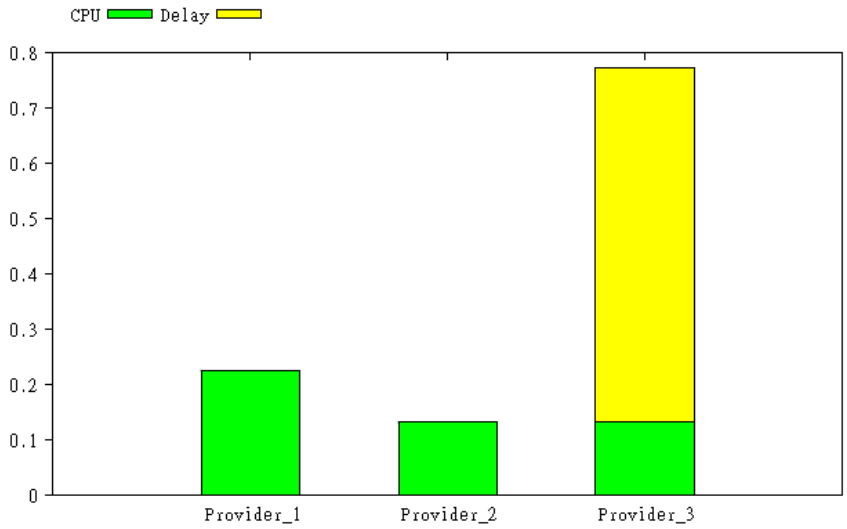

FIGURE 8: Les résultats de la recherche sémantique -

vice qui peuvent répondre à cette requête. Nous avons modélisé les fonctions d'utilité des trois critère négocié avec la forme linéaire, la forme exponentielle et les formes que nous avons proposé dans le chapitre précédant. Puis, nous avons calculé l'utilité globale attribuée à chaque fournisseur en utilisant les différentes formes. Nous remarquons que l'utilité globale attribuée à chaque fournisseur diffère selon la forme utilisée. Ainsi, le choix du meilleur fournisseur diffère selon la méthode utilisée. En utilisant les forme exponentielle et linéaire, le choix porte sur un fournisseur qui a une bonne bande passante mais qui ne satisfait pas les attentes de l'utilisateur en terme de temps de réponse. La compensation entre les utilités de ces deux critères entraine un mauvais choix de fournisseur. En utilisant les utilités que nous avons proposés, nous choisissons le fournisseur qui satisfait tout les critères négociés. Ainsi, nous satisfaisons les attentes des utilisateurs.

Évaluation des fonctions d'utilité non fonctionnelles Les critères non fonctionnels sont des critères non mesurable qui ne peuvent pas être directement représentés par des fonctions d'utilité. Par conséquent, des critères non fonctionnels comme la fiabilité et la réputation des fournisseurs ne sont pas pris en compte dans le choix du meilleur fournisseur. Ces critères

sont très important et doivent être inclus dans le processus de prise de décision et dans le choix du fournisseur de services. Afin de prendre en compte des critère non fonctionnels, nous proposons dans ce travail des méthodes pour calculer l'utilité de ces critère. Dans ce paragraphe, nous proposons des exemples numériques qui permettent de calculer les fonctions d'utilité du critère "disaster recovery" représentant le plan de reprise après un sinistre et du critère "Trust" représentant la réputation d'un fournisseur de service.

Contents

List of Figures

LIST OF FIGURES

LIST OF FIGURES

List of Tables

LIST OF TABLES

1

Introduction

1.1 Context

Cloud Computing is one of the hottest topics in IT as it fundamentally changes the ways institutions and companies are managing their computing needs. Cloud Computing has sprung as a new paradigm, for both enterprises and scientific applications development, managing and delivering services over the internet. So, what is cloud computing? Is it a technology recently invented?

The general idea behind this technology trace its roots back to the 1950s when large-scale mainframes were made available to schools and corporations. Because of the cost of buying and maintaining mainframes, it wasn't possible to purvey a mainframe to each user. So, multiple users were enabled to access the same mainframe via "dumb terminals". From here emerges the idea of sharing access to the same data storage layer and CPU power from any station. Later, around 1970, emerges the concept of virtual machines (VMs). VM is a tightly isolated software container with an operating system and application inside. It enables different operating systems to run in the same computer at the same time. The VM operating system took the 1950s shared access mainframe to the next level, permitting multiple distinct computing environments to reside on one physical environment.

In the 1990s, telecommunications companies were able to provide users with shared access to the same physical infrastructure. Hence, they started offering virtualized private network connections.

In 1999, the arrival of Salesforce.com pioneered the concept of delivering enterprise

applications via a simple website. Then the idea of delivering applications over the internet was reproduced by multiple providers like Amazon Web Services in 2002 and Elastic Compute cloud (EC2) in 2006.

Nowadays, cloud computing is the solution to the problem of how the Internet can help improve business technology. It emerges as a new computing paradigm which provides a large scale service provisioning. It is a framework delivering virtualized infrastructure resources as a service through a public network which is internet. It offers various key advantages such as cost effectiveness, information access from anywhere, quick deployment, almost unlimited Storage, backup and recovery capabilities, and flexibility to scale up and down, and it assures several benefits such as scalability, elasticity, reliability and data management as illustrated by figure 1.1.

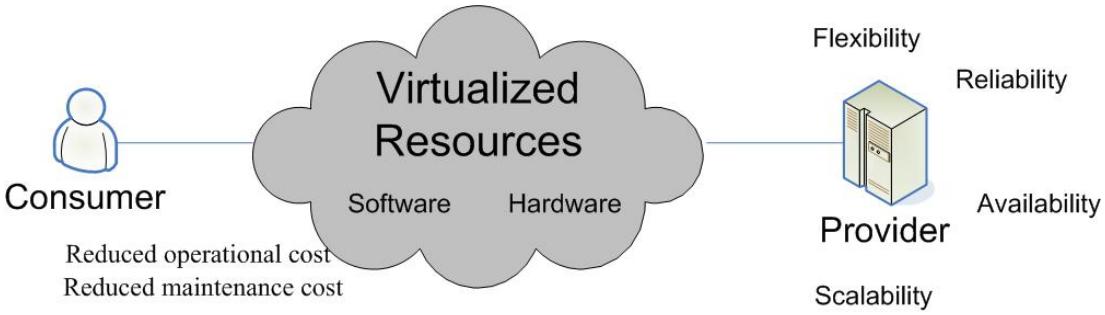

Figure 1.1: The cloud computing environment -

Cloud computing proved its usefulness for enterprises and markets which encourage a multitude of providers to offer real business solutions based on the cloud concept. With the rapid proliferation of cloud services, it is now difficult to know which ones are a good fit for a company's needs.

1.2 Problem statement

The growing popularity and adoption of Cloud computing solutions has attracted many customers from different natures. But face to the diversity and the heterogeneity of cloud service providers, the customers find a difficulty when selecting their best fitting Cloud provider. Indeed, the first problem faced by customers is the heterogeneity problem which consists in the diversity of existing ways for describing services. Actually, each provider establishes the definitions and parameters for its cloud offers. Hence, many

providers may use the same term to define completely different services (hybrid cloud is one example), making it difficult to compare offers (2).

Face to this heterogeneous cloud environment, customers may choose to contact only the providers using the same vocabulary. This could help them in avoiding ambiguity and misunderstanding, but, it ties them to particular providers which don't offer necessary the best solution for them. To avoid to choose unsuitable providers, customers have to work on understanding, analysing and summarising the different provider's proposal to be able to compare them which is a hard and time consuming task for them.

Moreover, to choose the right cloud provider, customers need also to compare the provider's quality of service parameters e.g., response time, bandwidth, reputation in the market, reliability .... Given the large number and variations of variables involved, it is hard to find a provider that offers good performance for all criteria. Therefore, we have to select the provider that proposes the best trade-off. This could be a difficult task especially involving not only functional criteria but also non functional criteria.

Selecting the right cloud service provider is a difficult process, but it is not the latest difficulty faced by the consumers. Indeed, after negotiating an agreement, cloud service providers don't offer any performance guaranty and leave the burden of detecting violations to the customers (3). Hence, throughout the execution of the service on the provider's resources, customers have to wary about the eventual violations of the agreement and impose the necessary penalties to the provider.

1.3 Contributions

To address the issues cited previously, an effective management of the relationship between providers and consumers is required for the delivery of cloud computing. In this work, we propose a third party entity that negotiates the relationships between cloud providers and cloud consumers called *cloud broker*. Our proposed cloud broker assists users in finding the right cloud provider, establishing a service level agreement and verifying that the provider honours the agreement terms. It aims to ensure:

1. the interoperability between cloud providers and cloud consumers by introducing semantic annotations in the negotiation process;

2. an optimum user satisfaction by finding him the best trade-off between QoS parameters based on his own preferences;

3. the sustainability of user satisfaction by providing him tools to supervise easily the compliance of the provided services with his requirements.

1.4 Thesis structure

Chapter 1 begins by introducing the concept of cloud computing, its characteristics, its service models, its deployment models and its main management challenges. It continues with introducing semantic based management tools and their utilization for network management automation in general which could be an interesting solution for the automation of the cloud management process. Finally, it defines cloud broker concept which is a third-party entity managing the relationship between the cloud actors and negotiating service level agreements in the cloud. In Chapter 2, we present the cloud broker architecture we proposed, its components and its main features. We explained how we used ontologies and rules for managing the SLA contract between providers and consumers, and how semantic annotations could ensure interoperability between the different cloud actors. The use of a multi-attribute utility theory method to find the best cloud provider is presented in Chapter 3. In this chapter, we introduced also our proposed utility functions for calculating the user satisfaction degree. Finally, we evaluate in Chapter 4 the usefulness of the introduction of semantic annotations in discovering more cloud services and in providing more accurate results. We evaluate also the advantages of using the utility functions that we proposed in the best provider selection.

Chapter 2

State of the art on cloud management frameworks

2.1 Cloud Computing Overview

Cloud computing is the on-demand delivery of computing services over the internet or a company network, or both. cloud services allow individuals and businesses to provision computational, network and storage resources from any device connected to Internet. Provisioned resources are scattered at remote locations all over the world which gives users a wide range of choice. Used technology in cloud computing allows customers to use applications without installation and to access their personal files simply by connecting at any computer with internet access. Cloud services are also designed to work equally well with Linux, Mac, and Windows platforms. This technology, based on virtualization concepts, aims to guaranty high availability, scalability and reliability. In this report, we adopt the definition developed by the U.S. National Institute of Standards and Technology (NIST) (4) : " Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." The main features of a cloud environment include a reduced cost, an improved efficiency and a rapid deployment of services. Cloud computing can result in reduction in capital and operating expenses. It allows deployed systems to run on the latest platform and it enables changes since it allows people to try new tools. This cloud model promotes

availability, scalability and federation and is composed of five essential characteristics, three service models, and four deployment models.

2.1.1 Cloud Computing Characteristics

Cloud computing is a relatively new business model in the computing world. Understanding the characteristics of cloud computing will help highlighting its key advantages and understanding its utility. There are myriad variations on the definition of the cloud, but, we will consider the characteristics identified by NIST in their definition. Here, are the five main characteristics that cloud computing offers businesses today.

- **On-demand self-service**: Computer services such as email, applications, network or server service are provisioned automatically without requiring human interaction from a cloud host provider. Typically, consumers are billed with a monthly subscription or a pay-for-what-you-use scenario. Terms of subscriptions and payments will vary with each software provider.

- **Broad network access**: Cloud capabilities are available over the network and accessed through a simple online access point such as mobile phones, tablets, laptops, and workstations.

- **Resource pooling**: The providers computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter). Examples of resources include storage, processing, memory, and network bandwidth.

- **Rapid elasticity**: Cloud services can be rapidly and elastically provisioned, in some cases automatically, to quickly scale out and rapidly released to quickly scale in. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time.

- **Measurable service - payment pay-per-use**: Going back to the affordable nature of the cloud, you only pay for what you use. Cloud computing resource usage can be measured, controlled, and reported providing transparency for both the provider and consumer of the utilised service. Cloud computing services use a metering capability which enables to control and optimise resource use. This implies that just like air time, electricity or municipality water IT services are charged per usage metrics pay per use. The more you utilise the higher the bill. Just as utility companies sell power to subscribers, and telephone companies sell voice and data services, IT services such as network security management, data center hosting or even departmental billing can now be easily delivered as a contractual service.

2.1.2 Cloud Computing Service Models

There are three basic kinds of cloud service models. Each share similarities but have their own distinct differences as well. These service models are Software-as-a-Service, Platform-as-a-Service and Infrastructure-as-a-Service. It helps to think of these services in layers as depisted by figure 2.1.

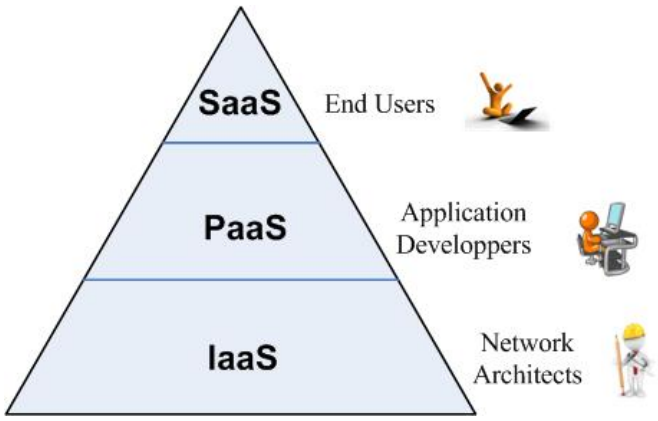

Figure 2.1: Cloud Computing Service Models -

Cloud computing services can be categorized into:

- **Software-as-a-Service (SaaS)**: also known as a software on demand. SaaS is a software distribution model in which applications are hosted by a vendor or service provider and made it available to users over the Internet. SaaS offers

already created applications running on a cloud infrastructure. Such applications are exploited is different domains like business domain, financial services, human resources, management etc. This model is the highest level at which cloud can be used by the customers, it eliminates the need to install hardware on the customers local computers and it applies updates without customer intervention. Examples of SaaS providers are: Freshdesk (5), NetSuite(6), Zoho (7), Bloomfire (8) and GHG Corporation (9), among others.

- **Platform-as-a-Service (PaaS)**:provides computing platforms which typically includes operating system, programming language execution environment, database, web server etc. Platform as a Service (PaaS) offers the possibility to exploit clouds in a different manner than using the virtualized infrastructure. It provides a development platform as a service for developers where applications are developed using a set of programming languages and tools. These services may include development, integration, testing or resources storage to complete the life-cycle of services. Examples of PaaS providers are: Google App Engine (10), AWS Elastic Beanstalk (11), cloudmapreduce (12) and Force.com (13), among others.

- **Infrastructure-as-a-Service (IaaS)**: IaaS (14) allows managing a large set of computing resources and provides its customers with physical or virtual computing infrastructure, including storage, hardware, servers and networking components. Resources like virtual-machine disk image library, block and file-based storage, firewalls, load balancers, IP addresses, virtual local area networks are rent from IaaS providers, then reconfigured to install IaaS customers applications on them. Scale up is assured by requesting more servers and reconfiguring the load balancer without purchasing more hardware and scale down is possible at any time by reconfiguring the infrastructure. Examples of IaaS providers are: Amazon E2C (15), GoGrid (16), Rackspace (17) and Windows Azure(18), among others.

2.1.3 Cloud Computing Deployment Models

Cloud services can be deployed in different ways, depending on the organizational structure and the provisioning location. Tree deployment models are usually distinguished, namely public, private and hybrid cloud service usage as illustrated in Figure 2.2.

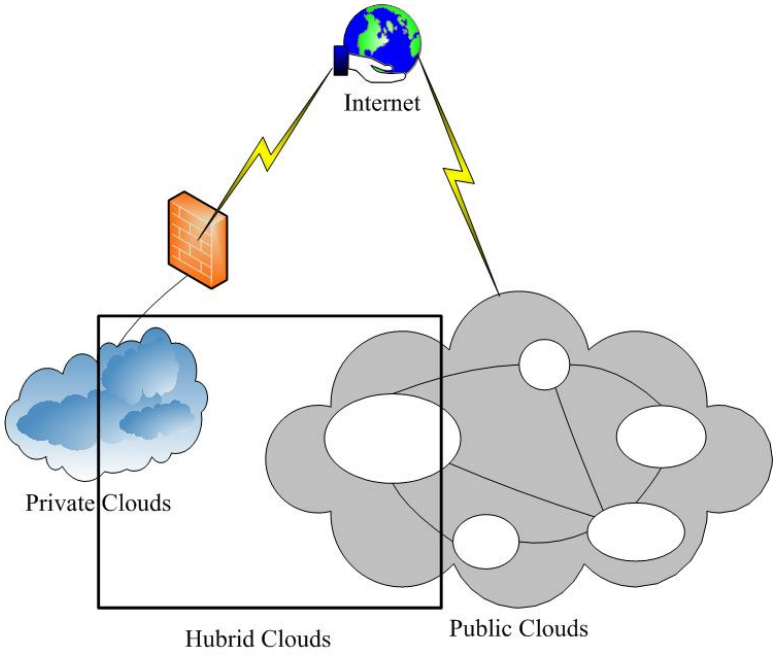

Figure 2.2: Cloud Computing Deployment Models -

- **Public cloud** refers to a set of computer and network resources characterized by a public availability of service offering and a public network that is used to communicate with the cloud service. Application and data are stored in the providers data center. Windows Azure Platform by Microsoft, AWS by Amazon, AppEngine and Gmail by Google, etc. are all examples of public cloud services. Public clouds are perfect for organizations looking to expand their testing or development environment. However, Customers who possess sensitive data and application normally do not feel comfortable using public cloud due to privacy, policy, and security concerns.

- **Private cloud** is dedicated to a single organisation provided with great control and privacy. The chief advantage of these systems is that hardware is locally managed, so, hosting organisation retains full control over corporate data, security guidelines, and system performance. In contrast, private cloud offerings are usually not as large-scale as public cloud offerings resulting in worse economies of scale.

- **Hybrid cloud** The idea behind hybrid clouds is that businesses can use them to

take advantage of the scalability and cost-effectiveness offered by the public cloud computing environment without exposing mission-critical applications and data to the vulnerabilities associated with the public cloud option. The deployment of an hybrid cloud computing system requires two or more clouds (private or public)that remain unique entities but are bound together by standardized or proprietary technology. With the benefits derived from both deployment models, the hybrid model solution has become more popular nowadays.

2.2 Cloud Computing Management Challenges

Cloud computing will play a major role in the future Internet of Services by offering on-demand access to shared resources and services. However, it must address several technology challenges (19, 20) to accomplish its business objectives. In order to better coordinate and accelerate standardisation in the field of cloud computing at the national and the European levels, the Federal Ministry of Economics and Technology has engaged Booz and Company to conduct a study defining cloud computing challenges (21). Nine especially relevant challenges have been identified which cover both the perspectives of providers and users and overarching interests. At a second level, they are subdivided again into 19 further subcategories. Defined cloud challenges are depicted by figure 2.3. A key challenge IaaS providers face when building a cloud infrastructure is

1	**Efficiency of service provisioning**
a	Usage of development tools & components
b	Creation of scalable architectures
c	Resource management & flexibility
d	Availability of services
2	**Effectiveness of service usage and control**
a	Contracts incl. questions of liability
b	Control of services by users
c	Governance/escalation mechanisms
3	**Transparency of service delivery and billing**
a	Billing incl. license management
b	Quality assurance and monitoring SLA
c	Type and location of data processing

4	**Information security**
a	Identity & rights management
b	Privacy & integrity
c	Access control, logging, attack prevention
d	Verification & certification
5	**Data privacy**
6	**Interoperability**
a	Migration in the/out of the Cloud
b	Ability to integrate into on-premise IT
c	Cloud federation
7	**Portability between providers**
a	Service portability
b	Data portability
8	**Ensuring fair competition in the market**
9	**Compliance with regulatory requirements**

Figure 2.3: Cloud challenges as introduced by Booz and Company -

managing physical and virtual resources, namely servers, storage, and networks. The

orchestration of resources must be performed in a way to rapidly and dynamically provision resources to applications. In the next section, we discuss the most common management challenges that emerge in the cloud computing domain.

2.2.1 Data Management

Cloud storage is a model of networked online storage where data is placed on multiple virtual servers, generally hosted by third parties; rather than being hosted on dedicated servers. Hosting companies operate large data centers. The data center operators, in the background, virtualize the resources according to the requirements of the customer and expose them as storage pools, which the customers can themselves use to store files or data objects. This way, the resource may be span physically across multiple servers. Clouds have a single point of access for all computing requests so consumers will have access to data from any point, on demand. From the business point of view, Cloud infrastructures aim to provide robustness and availability at any time. Scalable and consistent data management is a challenge that has confronted the database research community for more than two decades (22, 23). Moreover, there are limitations on the size of the objects that can be stored, which can create some complications in the development process. The fine-grain access is another issue since IaaS provides just simple mechanisms like get and put for managing the data, and these operations cannot access just small parts.

2.2.2 Virtual Machine Management

The use of virtualisation technologies enable service providers to optimise the use of resources. Indeed, until recently, operating systems managed the allocation of physical resources, such as CPU time, main memory, disk space and network bandwidth to applications. Virtualisation infrastructures, such as Xen and VMWare. are changing this by introducing a layer of abstraction known as a hypervisor. Dynamic management of virtualized application environments has become an active area of research in the cloud computing paradigm, especially with recent virtualization capabilities that allow live sessions to be moved transparently between servers (24, 25). Cost of resources varies significantly depending on configuration for using them. Hence efficient management of resources is of prime interest to both cloud providers and cloud users. Performances of virtual machines depends also on the resources usage of other virtual machines using

the same physical resources. Thus, the need of a scheduler orchestrating the virtual machine placement within the cloud. The scheduling algorithms are divided into static scheduling and dynamic scheduling. Static approaches are based on the prediction of the process behaviour then they guaranty response times. Dynamic scheduling are more flexible since virtual machines are automatically turned on or off based on demand and threshold policies. The architecture of schedulers could be categorised into centralized approaches and decentralized ones. There are several virtual machines schedulers such as Snooze, Entropy, open Nebula scheduler and Haizea (26).

2.2.3 Load Balancing

Load Balancing (27) is used to distribute workload across one or multiple servers, network interfaces, hard drives, or other computing resources. This method is used to optimize resource usage by avoiding overload of any one of the resources, minimize response times and maximize throughput. Load balancers are typically used within data centers as part of the overall application delivery controllers. They represent a traffic management solution that functions at the DNS layer. They are introduced to overcome some of the significant networking challenges associated with providing fully elastic and scalable self-service capabilities when delivering application services in service provider environments. Load balancing is especially useful for networks where it is difficult to predict the number of requests that will be issued to a server. It can be implemented with dedicated hardware or software, or a combination of both. Using multiple components with load balancing instead of a single component may increase reliability through redundancy. Typically, load balancing is the main reason for computer server clustering. Load balancing has become a challenging research area for efficient operations in a cloud environment. Diverse static and dynamic algorithms are proposed to resolve the issue of load balancing and task scheduling presenting many advantages and disadvantages.

2.2.4 Security

Cloud computing opens up a new world of opportunities for businesses, but mixed in with these opportunities are numerous security challenges that need to be considered and addressed prior to committing to a cloud computing strategy. Because of the cloud service models employed, cloud computing presents more risks than other IT environments. Security (28, 29) and privacy are still cited by many organisms as the

top inhibitors of cloud services adoption. The cloud, especially public clouds, highlights new and significant security concerns for companies that are accustomed to hosting their data and applications within their own servers. Cloud computing security challenges fall into three broad categories:

- Data Protection: Securing your data both at rest and in transit
- User Authentication: Limiting access to data and monitoring who accesses the data
- Disaster and Data Breach Contingency Planning

2.2.5 Service Level Agreement

Service Level Agreements (SLAs) (30) has been used since 1980s in a variety of fields, so, most of the available definitions are contextual. In the area of cloud computing, SLAs are agreements between a service provider and another party such as a service consumer, a broker agent or a monitoring agent. It is a formal contract used to guaranty that consumer's service quality expectation can be achieved and it provides a unique combination of business-driven application scenarios and advanced research in the area of service-level agreements for clouds and service-oriented infrastructures. SLAs are offered by IaaS providers to express their commitment to delivery of a certain QoS. An SLA usually include availability and performance guarantees. Additionally, SLA metrics could be classified by different categories which are compute SLAs, Network SLAs and Storage SLAs. metrics must be agreed upon by all parties as well as penalties for violating these expectations. Most IaaS providers focus their SLA terms on availability guarantees, specifying the minimum percentage of time the system will be available during a certain period.

Actually, there is no standard to express cloud SLA. Each provider describes its service level agreements in its own way. Thus, several languages for specifying SLA have been proposed:

The Web Service Level Agreement (WSLA) The Web Service Level Agreement (WSLA) (31) provides a framework for defining and monitoring service level agreement (SLA) for Web services. It could be used also in other domains such as business process

and service management, or the management of networks, systems and applications in general. It consists of a formal and extensible language, designed to specify SLAs in a flexible and individualized way. It is XML-based used by both service providers and consumers. Since SLA specification is determined, the WSLA monitoring services are automatically configured to enforce the SLA. However, The WSLA specifications don't foresee creation and form of a WSLA template. Though a WSLA document without specified parties may potentially be used as a template, the specifications don't support ranges of values for obligations. An implementation of the WSLA framework, termed SLA Compliance Monitor, is publicly available as part of the IBM Web Services Toolkit. Figure 2.4 shows a short recap of WSLA.

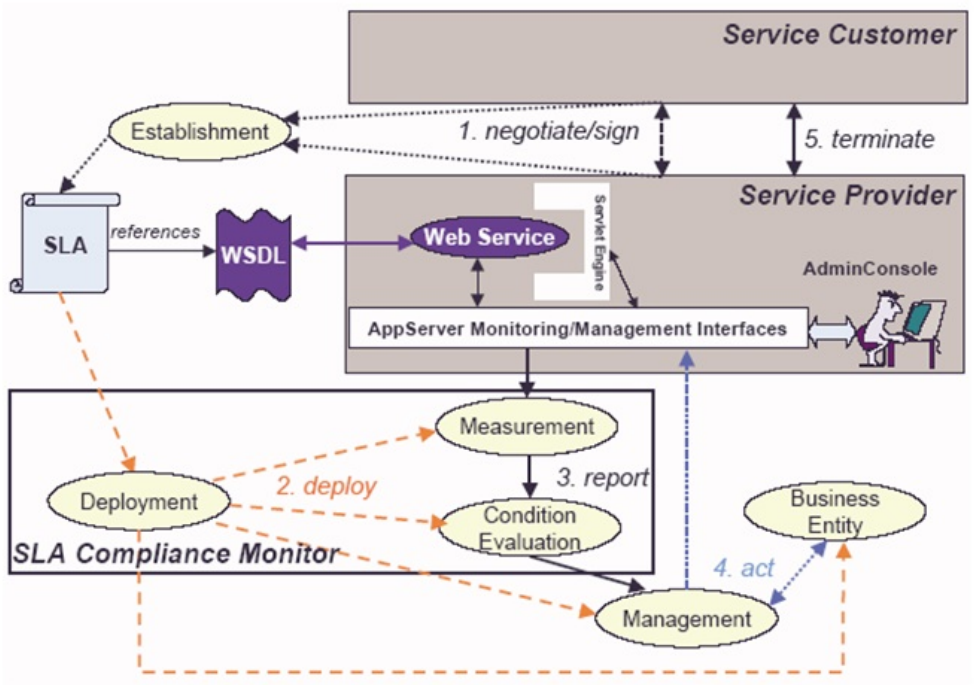

Figure 2.4: WSLA Services and their interactions (32) -

SLAng SLAng (33) is a language for concrete service-level agreements currently providing support for ASP SLAs. It provides a format for the description of QoS properties using an EMOF metamodel, with embedded OCL constraints and natural-language documentation in English. SLAng defines QoS targets including reliability, timeliness,

availability, data currency, data recovery. The application of these constraints can be varied according to the level of abstraction at which the system is described. SLAng is designed so that all SLAs expressed in the language are monitorable. This makes it extremely precise as well as being understandable. It also includes semantics for administration, which is the process whereby the parties to the SLA agree what penalties are to be paid. It expresses constraints on the accuracy of reports used in administration that are approximately monitorable. However, it is not enough expressive to represent the QoS parameters included in SLA. SLAng defines seven different types of SLA. They regulate the possible agreements between the different types of parties identified in the SLAng model which are Application, Web Service, Component, Container, Storage and Network. Vertical and Horizontal SLAs can be contracted between pairs of them. The Vertical SLAs are:

- Application: between applications/web-services and components.
- Hosting: between container and component providers.
- Persistence: between a container provider and an SSP.
- Communication: between container and network providers.

The Horizontal SLAs are:

- Service: between component and web service providers
- Container: between container providers
- Networking: between network providers

WS-Agreement The Web Services Agreement (34) specification constitutes a normative language to formulate Service Level Agreements and a basic protocol to expose service-level descriptions, validate service-level requests, and come to an agreement. This Java framework, proposed by the OGF WS-Agreement standard, defines a tool to create and manage service level agreements in distributed systems. The WS-Agreement specification defines two separate schemata, the agreement schema and the agreement state schema. The agreement schema that defines the WS-Agreement core data types and the agreement state schema that includes the data types for the dynamic agreement

monitoring, namely the agreement states, service term states and guarantee term states. WS-Agreement allows a service provider and a service consumer to decide whether to accept or reject a service offer. Although this approach is sufficient for a number of use cases, others exist with requirements for multi-step negotiation or the adaptation of an existing agreement.

Generally, each provider decide about the SLA language used to communicate with the consumers. Starting from the selected language, he can define an SLA template in which he specifies for example the quality of service parameters. The definition of the QoS parameters may differ from one provider to another i.e., two different providers may refer to different concepts by the same QoS parameter. This can lead to a misunderstanding between providers and consumers resulting in a bad discovery and selection services regarding the consumer's requirements.

Next section targets works dealing with service discovery and selection in the cloud.

2.3 Service Discovery and Selection

In order to reduce the heterogeneity between various equivalent services, several works have proposed to use ontologies and semantic annotations. In the following, we will introduce semantic based management tools such as ontologies and declarative programming and we will present relevant works that used ontologies for cloud service discovery.

2.3.1 Semantic based management tools

2.3.1.1 Semantic Web architectures

Semantic Web architecture (35) is the next generation in information architecture. Semantic Web enables data to be presented in an efficient way to be understood and used by machines without human intervention. The most well-known versions of the layered architecture that exist within literature have been introduced by BernersLee (36). Berners-Lee proposed four versions of Semantic Web architecture as depicted by figure 2.5. The fist one in 2000, the second in 2003 as part of a presentation at the SIIA Summit, the third and fourth versions in 2005 and 2006 respectively. The main difference between those versions consists in the positioning and interaction between

ontology and rules layers. In fact, Semantic technologies represent meaning using ontologies and provide reasoning through the relationships, rules, logic, and conditions represented in those ontologies.

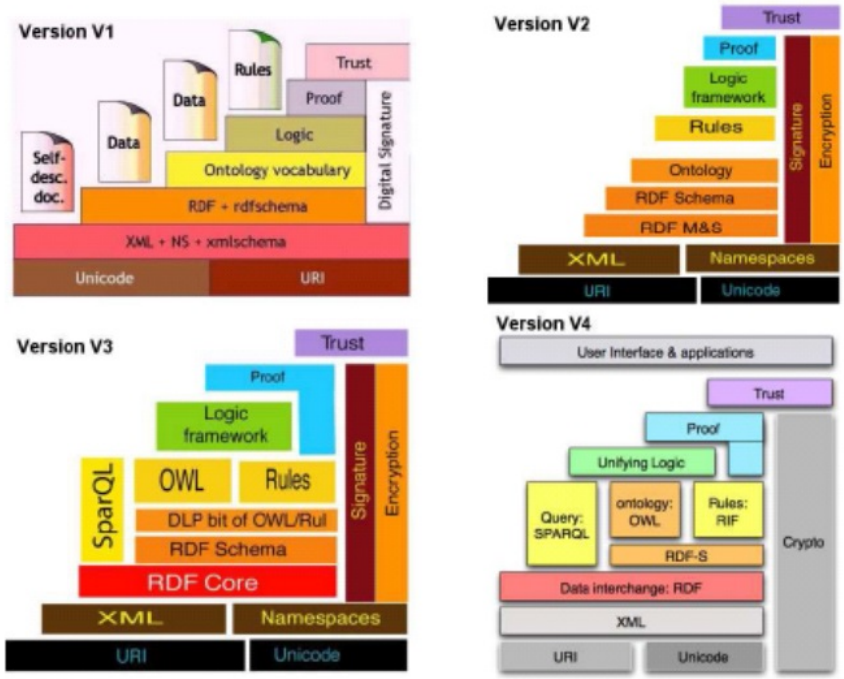

Figure 2.5: Semantic Web Architectures (35) -

2.3.1.2 Ontology

The use of ontologies (37) has shown its benefits in many domains especially artificial intelligence, knowledge representation and inductive reasoning. Some generic ontologies have been designed to provide a framework for building service level agreements. They contribute essentially to identify concepts in the SLA domain. Moreover, the combination of ontologies with rule-based knowledges (38) is recommended for many interesting semantic web tasks.

Ontology can be defined as a formal, explicit specification of the terms of a shared conceptualization and relations among them. It is not a simple hierarchical structure of items but a logical description of concepts, relations among them and individual insertion. It has been introduced for explicating semantics of formal languages used in

knowledge sharing. One of the main advantages of employing ontologies expressed in logic-based formalism such as OWL is being able to check the logical consistency of our model using reasoners. Standard reasoner services are: consistency checking, subsumption checking, equivalence checking and instantiation checking. Such mechanisms facilitate automation, especially consistency checking mechanism, avoiding the creation of classes that doesn't really make sense. Reasoners help also inferring ontologies by deriving further expressions and relations that are not explicitly contained within the ontology.

Web Ontology language (OWL) is a knowledge language constructed over the RDF layer in the semantic web architecture, specified by the W3C consortium. OWL became the most used language for ontology description. OWL as a language has three sub languages: OWL-full, OWL-DL and OWL-lite. OWL-Full, as its name indicates, is a full version of OWL language. It allows a maximum expressiveness but may be undecidable and hence difficult to reason over. OWL-DL defines a decidable OWL language corresponding to the logic description SHOIN(D). OWL-Lite is a simple version of OWL corresponding to the logic description SHIF. It is decidable too. OWL-Lite is only useful for customers requiring a hierarchical classification of concepts supported by some simple constraints. This version of OWL can not be adopted for Service Level agreement specification requiring more expressiveness. OWL-Full can't be retained too because of its undecidability, while SLA specification is harnessed in automated systems so its ontology needs to be reasoned over. Ultimately, OWL-DL seems to be the best OWL version for SLA description due to its richness and its decidability.

2.3.1.3 Declarative programming

In contrast to conventional imperative programming languages such as Java or C++ specifying a series of instructions to be executed, declarative rule languages based on logic programming (LP) states what computer should do rather than how it should do it (39). It declares a set of rules about what outputs should result from which input. Declarative programming adduces a high level of flexibility. In fact, rule languages develop flexible applications on a high abstraction level. They are seen as self-contained knowledge units that involves some form of reasoning. Rules are classified as three types depending on their purpose:

- Deductive rules or derivation rules: used to derive implicit facts by reasoning on existing knowledge. Deductive rules inference conditions to get conclusions.
- Normative rules or integrity rules or structural rules in the business rule community: express integrity constraints ensuring consistency of data or knowledge bases, so they should be fulfilled throughout the systems life cycle.
- Reactive rules or active rules or dynamic rules or reaction rules: describe the reactive behavior of a system. This type of rules is used to update databases. There are subdivides into kinds of reaction rules:
  - Productive rules: rules of the form if Condition then Action, used in logical applications especially to manage the state of web nodes.
  - Event-Condition-Action rules (ECA rules): rules of the form on Event if Condition then Action, used to manage distributed systems relying on event based communication. Many ECA rule languages have been implemented such as Xchange, Drools flow, Alfresco, Plone and Sitecore CMS and others.

Many real-life problems cannot be represented only using ontologies and cannot be solved using barely ontological reasoning. Besides, the logic programming provides a very expressive formal language, however it requires domain knowledge to be encoded as a part of logic programs. Several works have focused on the capabilities of combining ontologies, based on description logics and rules, based on logic programming to enhance management automation.

2.3.2 Combining ontologies with rules

A combination of description logics and horn rules could be imagined as a possible approach for enhancing management. In the following, we will present some existing proposals using reasoning with rules and ontologies for management automation.

Stoutenburg et al (40) proposed SWORIER system acronym for Semantic Web Ontologies and Rules for Interoperability for Efficient Reasoning. The system's objective is to semantically enhance its ability to react quickly to unexpected changes. They proposed ontology and rules design and a framework gathering OWL, ruleML and SWRL technologies. This framework translates all the knowledge base, via an XSLT translator, to Prolog language that can be queried by users as depicted by figure 2.6

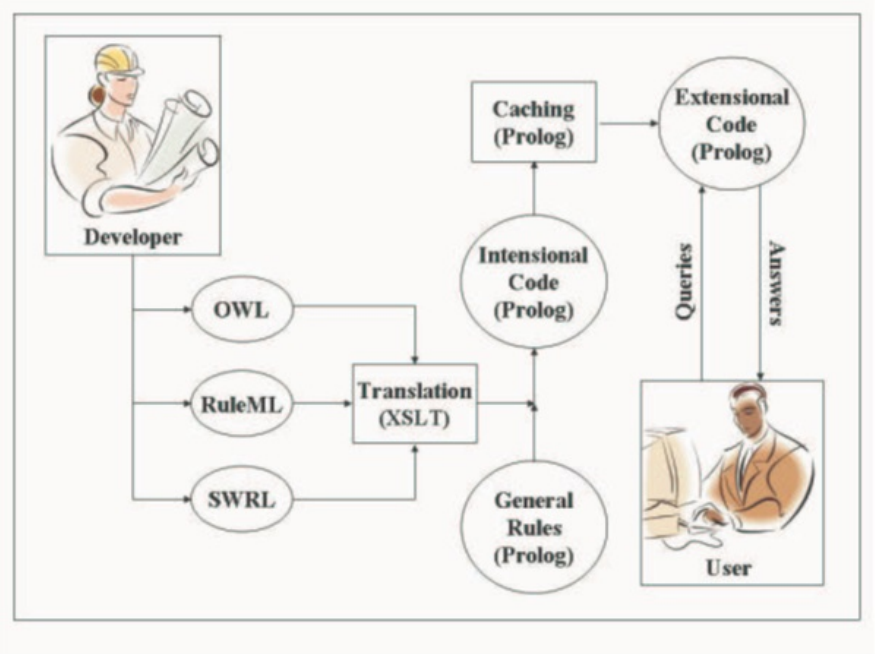

Figure 2.6: High-level approach to translating OWL, SWRL, and RuleML to Prolog (40) -

Authors proposed a military use case that alerts when detecting stationed enemy snipers. Ontologies and rules are used to model this use case and how they supported rapid, enterprise integration. Ontologies were applied in an overall framework for situational awareness. Conditions, alerts and recommendations are generated through the application of rules. Rules operated over the ontologies as part of an integrated knowledge base that could be queried dynamically. The proposed architecture is implemented using different components including a google earth client, a prolog reasoner, a knowledge base, a situational awareness service, an event mediation services, some Adaptors and a message simulator. To integrate those different components, authors use an Enterprise Service Bus (ESB). The ESB provides an abstraction layer over disparate messaging technologies, allowing interaction between components with minimal code development. Mule ESB has been selected to integrate sources for satellite information and other events by creating a Mule endpoint. It provides support for transport and transformation of publisher/subscriber pairs, applying the XSLTs of the Adaptors when appropriate. The architecture of the proposed system and the ESB role are illustrated by figure 2.7.

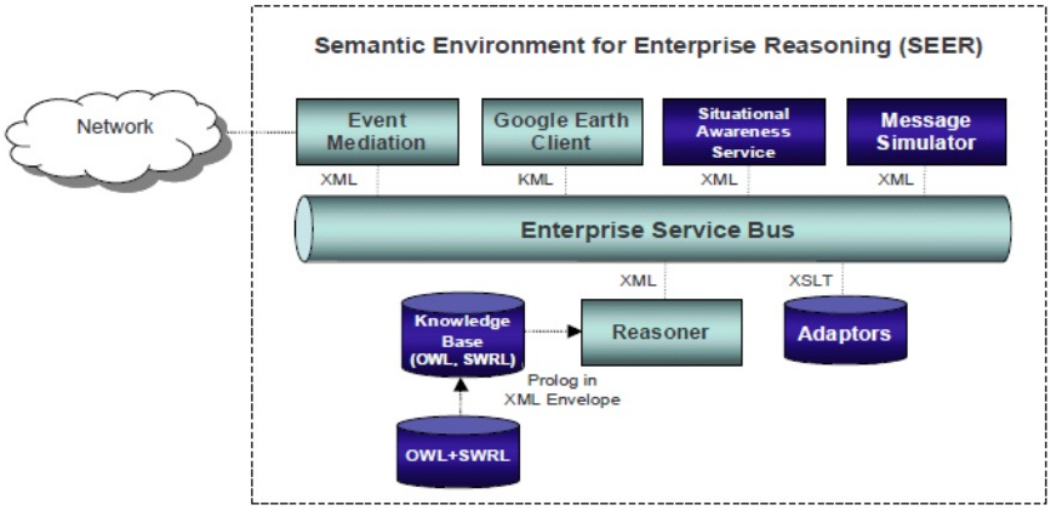

Figure 2.7: Architecture of the Semantic Environment for Enterprise Reasoning (SEER) (40) -

An integration of Ontology-based and policy-based Network management for automation is a second idea presented by Xiao and all (41). Its principal is to combine network management information and network management policies in the same model by the use of ontologies. Since ontologies are not enough rich to express restrictions of network policies, authors expresses Management Information Base (MIP) and Policy Information Base (PIB) using OWL standard language for ontologies together with SWRL rule language. Corresponding actions, invoked by those policies, can then be defined in the form of network management services described by OWL-S. So:

- OWL: define network management information,
- SWRL: define behavior definitions. SWRL provides a way to express those implicit restrictions on network management information in a formal explicit way
- OWL-S: define service definitions, OWL-S defines services corresponding to network management actions, which are invoked if the given condition defined in SWRL occurs.

The application of OWL, SWRL and OWL-S to describe MIP and PIB are illustrated by figure 2.8.

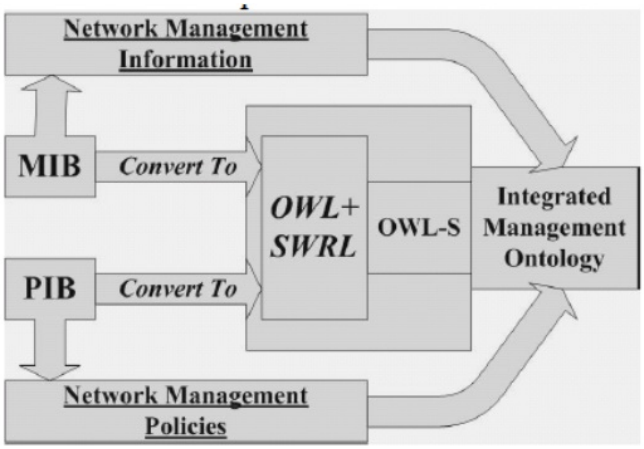

Figure 2.8: Application of OWL+SWRL and OWL-S to describe MIB and PIB (41) -

Authors have defined three steps to implement the integrated network management for automation:

- Step 1: Use an ontology tool, such as Protégé-OWL editor, to convert PIB and MIB to OWL+SWRL network management ontology manually, semi automatically or automatically, depending on the function level of the chosen tool.
- Step 2: According to the behaviours defined by SWRL for both network management information and policies, corresponding actions can then be defined in the form of network management services described by OWL-S, still with an ontology tool. Once the behaviour defined in this unified ontology is invoked, a corresponding action is performed according to the OWL-S definition.
- Step 3: Put all the polices contained in the unified network management ontology to the policy repository, which is used to store the policies generated by network management tool, so that they can be used by the IETF PBNM architecture.

With the proliferation of cloud computing solutions, many cloud providers emerges to propose different cloud services. We notice a huge semantic heterogeneity in the proposed services witch incited many research to use ontologies. Ontologies have been used to solve four main issues in the cloud: cloud resources and services description, cloud security, cloud interoperability and cloud services discovery and selection (42). In the following, we will focus on cloud services discovery and selection related work.

2.3.3 Cloud services discovery ontologies

Several works defined ontologies to discover cloud services among which we can cite the work proposed by

Han et al (43) presenting a cloud service discovery system (CSDS) that helps cloud users in finding cloud services over the Internet. To enhance the performance of the CSDS, they introduce a cloud ontology representing the taxonomy of concepts of different cloud services. This ontology enables the CSDS to reason about the relation between and among cloud service concepts to determine the similarity between two services. Three kinds of reasoning methods are used: similarity reasoning, equivalent reasoning and numerical reasoning. With the use of cloud ontology, the CSDS is more efficient since it is more successful in locating cloud services and more likely to discover cloud services that meet consumers' requirements.

Zhang et al (44) argue that comparing manually the service configurations of different cloud providers is not easy because they use non standardized naming conventions, various formats and heterogeneous types and features of cloud services. To overcome this problem, they propose to formally capture the domain knowledge of services using semantic Web languages like the Resource Description Framework (RDF) and the Web Ontology Language (OWL). They defined an ontology that Identifies the most important concepts and relations of functional and non-functional configuration parameters of infrastructure services. This ontology facilitates the description of cloud infrastructure services; and through mappings from provider descriptions, facilitates the discovery of infrastructure services based on their functionality and Quality of Service (QoS) parameters.

Ma et al (45) aim to dynamically allocate cloud resources suitable for cloud user requirements. For this purpose, they designed an architecture for ontology-based resource management system for cloud computing. The proposed ontology is built based on cloud resource information and agreed SLAs. It is used to define concepts underlying the proposed system for cloud computing and describe their relations, and inference rules are defined to give semantic meanings to all cloud computing object information. The experimental results of this work have showed that the ontology-based resource management system improves the efficiency of resource management for cloud Computing when compared to the existing resource management algorithms.

Several efforts have been conducted to enhance cloud services discovery and selection. They have verified that the use of ontologies improves the discovery of more services suitable for the user request. However, the construction of the proposed ontologies does not comply with or take into account any standardization efforts proposed such as OCCI standard that defines how cloud service providers can provision their resources and services to end users which. This makes hard the mapping to the provider's format.

On the other hand, the existence of several equivalent services let arising the problematic of choosing the best one.

2.3.4 Cloud Service Selection

In order to select the best service, consumers have to define criteria that discriminate the different services. However, consumers may define a large set of criteria that should be satisfied in the choice of the best service. The resolution of this problem is handled by the decision making area.

"*Decision making is the study of identifying and choosing alternatives based on the values and preferences of the decision maker. Making a decision implies that there are alternative choices to be considered, and in such a case we want not only to identify as many of these alternatives as possible but to choose the one that best fits with our goals, objectives, desires, values, and so on.*" (46)

Hereafter, we present decision-making techniques and tools tending to support users in making decisions in the cloud domain.

Decision making protocol An interesting work to manage decision-making is proposed by Waters et al. It consists on suggesting a Decision-Acquisition System Based on a Common Decision-Exchange Protocol (47). Common Decision Exchange Protocol (CDEP) is an information model that contains the possibly interesting information about a decision scenario. It considers everything as resource and uses the Hyper-Text Transfer Protocol (HTTP) language to present all resources. This way, information is better shared to support better and faster decision making. CDEP Protocol could lead to semi-automation of certain decision-making processes. This protocol could be applied in different domain such as military domain, health care, emergency management ...

Semantic tags Kodeswaran et al (48) proposed a multi-tier system for managing multi-tier networks using semantic tags. They define a policy based network model that eases network management and automates network configuration. In this architecture, policies are classified as enterprise wide. They are distributed to various autonomous domains that are responsible for enforcement of those policies within that domain and all combined sub-domains. Authors proposed, among others, two interesting entities: The Policy Decision Point (PDP) acting as the decision making entity and the Policy Enforcement Point (PEP) responsible for enforcing the policies at the device level. PDP is the entity that is responsible for reasoning over the network traffic utilizing the content metadata, network state and other contextual information available to it and determining the policies that need to be enforced. It is responsible for reasoning over the policies (using its Configuration Reasoner). PEP executes commands received from PDP. PEP's main responsibilities and actions are:

- Request and store its configuration from the local-PDP that is responsible for this device.
- Delegate any policy decisions to the local-PDP by extracting content metadata from data packets and adding to this description, any additional information that may be useful to the local-PDP
- Report errors and status updates to the local-PDP

This protocol is tested using the NS2 simulator. The PDP was implemented as a Java process that received OWL streams from a client PEP (a network router within NS2), invoke the reasoner and send back the Tcl commands depending on the actions that needed to be invoked. The PEP , in this case is the NS2 simulator, would then execute the commands received from the PDP.

Heuristics Wu et al (49) proposed an automated SLA negotiation framework for cloud computing. This framework includes decision making heuristics to negotiate the SLA contract between cloud parties. Its main components are: Customer Agent (CA), Broker Coordinator Agent (BCA), Provider Agent (PA), IaaS Provider, SLA Generator, Directory, Policy Database (PD), and Knowledge Base (KB). The broker coordinator agent includes a decision making system which includes heuristics , considering factors

such as time, market constraints and trade-offs to satisfy the different objectives of cloud actors. It minimizes price and guaranteed QoS for consumers, maximizes the profit of the cloud provides by receiving as many requests as possible and maximize the profit of the broker from the margin between the customer's budget and the provider's negotiated price.

Multi-criteria Decision Making Since the main advantage of adopting a cloud technology is economic, most of utilized methods for taking decisions in a cloud context, limits the decision making to the relative cost of cloud resource leasing. Therefore, they might neglect crucial key factors influencing the Quality of the Service and the client satisfaction. To address this shortcoming, some works propose to employ a multi criteria decision making method which is a qualitative comparison approach to evaluate alternatives against dissimilar criteria. Multi-Criteria Decision-Making (MCDM) (50) (51) (52) is a sub-discipline of operations research that explicitly considers multiple criteria in decision-making environments. It is an alternative to classical optimization methods based on the definition of a single function, often expressed in economic term (monetary) and reflects the consideration of several criteria, often immeasurable. The interest of multi-criteria methods is to consider a set of criteria of different kind (expressed in different units) and its ability to judge accordingly different alternatives as illustrated by figure 2.9.

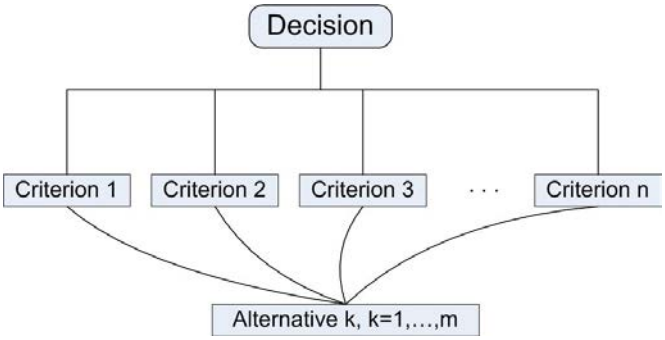

Figure 2.9: Multi criteria decision making (MCDM) Tree -

MCDM methods are gaining importance as potential tools for analysing complex real problems due to their inherent ability to judge a multitude of alternatives on different criteria for possible selection of the best suitable alternative(s). They interested some researchers in the field of cloud computing as a tool helping them to select the

best cloud service. For example, Menze et al (53) introduce a generic multi-criteria-based decision framework and an application for cloud computing, the Multi-Criteria Comparison Method for Cloud Computing $(MC^2)^2$. The framework and method allow organizations to determine what infrastructure best suits their needs by evaluating and ranking infrastructure alternatives using multiple criteria. Therefore, $((MC^2)^2)$ offers a way to differentiate infrastructures not only by costs, but also in terms of benefits, opportunities and risks. $((MC^2)^2)$ can be adapted to facilitate a wide array of decision-making scenarios within the domain of information technology infrastructures, depending on the criteria selected to support the framework. Ur-Rehman and all (54) introduced a methodology for selecting cloud services based on multiple criteria. They propose a formulation of the service selection problem into a generalized and abstract mathematical form. Then, they build a service selection method based on a comparison between the user requirement criteria vector and all service descriptor vectors. This method leads to the selection of the service having the corresponding descriptor vector that best matches the user requirement vector.

The over mentioned issues concentrate on multi criteria decision making but do not explain how those criteria under negotiation could be formulated in an SLA contract. Besides, they do not propose guarantees to ensure that the best choice founded by the multi-criteria method still satisfies the cloud consumer throughout the duration of the deployed service execution.
In order to overcome those difficulties, some works have proposed cloud broker architectures to provide cloud-users a unified and enhanced management interface to multiple cloud-providers. This will be the subject of the following section.

2.4 Cloud Broker architecture

2.4.1 Cloud Broker Definition

A cloud broker is a third-party individual or business negotiating relationships between providers selling services and consumers purchasing cloud computing services. The National Institute of Standards and Technology (NIST) defines the broker as (55) : "an entity that manages the use, performance and delivery of cloud services and negotiates relationships between cloud providers and cloud consumers".
Significant enhancement will be achieved in the future of cloud computing due to the

cloud broker notion since it provides an abstraction layer between providers and users. It guaranties an accurate negotiation contract leading to compute and storage services provisioning. NIST attributes three categories of services to the cloud broker: service intermediation, service aggregation and service arbitrage. Indeed, the cloud broker provides the following capabilities:

- Helping users determine the best framework for each individual need, based on a number of factors. This includes discovering suitable data sources for a given analysis scenario, selecting suitable computational resources and optimally mapping analysis jobs to compute resources.
- Providing a single interface for interaction with multiple clouds, hiding the complexity inherent in working with multiple providers.
- Monitoring and controlling clouds in a flexible way. The broker negotiates technical contracts, access data from local or remote data source, detects cloud failures and reacts in some appropriate way.
- Saving money and being more efficient: the cloud broker provides a cost-effective resource utilization.
- Reducing security risks: taking care of authentication, authorization and access control.

2.4.2 Cloud Broker related work

Jrad et al (56) proposed an SLA based service brokering in intercloud environment. They designed a high-level generic architecture by integrating several state of the art technologies and standards, illustrated by figure 2.10.

The main components of this architecture are:

- SLA Manager: responsible for SLA negotiation and provisioning.
- Monitoring and Discovery Manager: responsible for monitoring SLA metrics.
- Match Maker: responsible for selecting the best cloud providers using different matching algorithms.
- Deployment Manager: responsible for deploying the service on the selected provider.

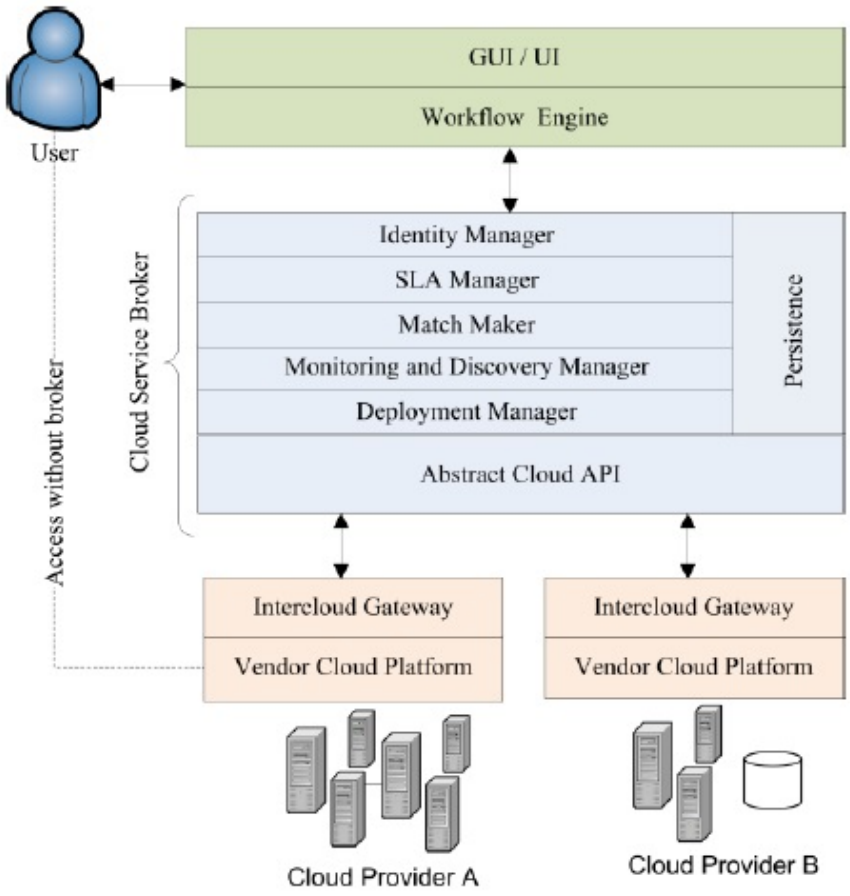

Figure 2.10: An SLA based service brokering in intercloud environment (56) -

- Identity Manager: responsible for user authentication, IDs and roles enforcements.
- Persistence: responsible for storeing broker specific data (e.g. monitoring, SLA templates and resources data).
- Abstract Cloud API: responsible for managing cloud resources on different cloud providers.
- Intercloud Gateway: responsible for the interaction through the standard cloud API with the broker.
- Vendor Cloud Platform: is the native cloud platform hosted by the cloud provider.

Pawluk et al (57) designed a cloud broker service they named STRATOS. It represents an initial step toward the automated cross-cloud resource provisioning and intercloud platform. This solution is responsible for solving the Resource Acquisition Decision (RAD) problem. It automates the decision at runtime rather than deployment time. It selects a set of configurations satisfying user objectives that are optimized using a multi criteria optimization formulation. In a first step, the cloud manager contacts the broker to instantiate the topology. It provides the broker with a topology document specifying the topology to be deployed on the cloud. It includes structural concerns (e.g., numbers of tiers in the application, numbers of nodes in each tier, etc.), monitoring directives (e.g., which metrics to monitor and how often), management directives (e.g., a set of models to control the elasticity policy of the application server tier at runtime), and the deployer's objectives. Afterword, the broker performs the initial RAD calculation, namely the most efficient allocation of resources across providers. The RAD problem is formulated as a multi-criteria optimisation problem. The broker requires two pieces of information from the deployer to solve any particular RAD problem: desired configuration and a set of objectives. A configuration is described by a list of properties where each property is a triple (name; value; unit). An objective represents a utility function calculated for a topology, with a configuration being a variable, e.g., cost of the topology can be viewed as an objective to be minimized.

To solve this problem, several research projects proposed brokering architectures. We could mention for example the

SLA@SOI (58) European project addresses the issues surrounding the implementation of automated SLA management solutions on Service Oriented Infrastructures (SOI)

and evaluates their effectiveness. In particular, the SLA@SOI framework has the responsibility of acting as a broker. Its SLA management layer supplies the user requests in the form of infrastructure SLA (iSLA) requests, then, selects the most appropriate provider based on the iSLA terms . This architecture aims essentially to offer a generic solution for SLA management that can: (1) support SLA management across multiple layers of a service-oriented infrastructure; (2) cover the complete SLA and service life cycle; and (3) be used in various industrial domains and use cases as mentioned in (59).

Optimis project (60) proposes a broker model involving the brokering of a federation of providers to propose an SLA-based tiered pricing model to the customers of the broker. This broker (61) aims to negotiate SLAs between consumers and providers and effectively match the requirements of cloud consumer with the provider's service. It aims also to maintain performance check on these SLAs and take actions against SLA violation. The Optimis broker uses the WS-agreement standard to negotiate SLA contracts and to perform a match between the requirements of the service provider and the functionalities provided by the various infrastructure providers.

The Contrail project (62) aims to design, implement, evaluate, and promote an open source system for cloud federations. It aims to provide the federation users the view of a single cloud. For this purpose, it propose a cloud broker solution that selects the best or most suitable public cloud. This selection is based on the cheapest or most security conscious or in a specific location i.e., whatever their specific needs are for that piece of work.

The Open Source API and platform for multiple clouds (mOSAIC) project (63) aims at creating an open source API and a platform to allow using multiple cloud services at once. It offers also a brokerage system to support the decision of cloud service provider selection at the deployment stage. There is a cloud agency module that plays the role of broker in the mOSAIC's architecture. Its basic services are (1) the negotiation of SLAs, (2) the deployment of cloud services, and (3) the discovery and brokering of cloud services. To fulfil these goals, the mOSAIC project defined an ontology able to describe services and their (wrapped) interfaces (64). The defined ontology describes services at the three delivery models of cloud computing (i.e., IaaS, PaaS and SaaS) and enables intelligent selection of services, with automation of different tasks, including service discovery, mediation, invocation, or composition.

The Open Source Cloud Broker "CompatibleOne" (65, 66) provides interoperable middleware for the description and federation of heterogeneous clouds and resources provisioned by different cloud providers. It assists cloud end users in their providers choice and it allows them to avoid vendor lock in, enforce SLAs and reduce costs. The CompatibleOne broker is based on open standards, mainly CDMI and OCCI, and uses new defined object-based description models such as the compatibleOne resource description system .

These over-mentioned solutions still present some deficiencies and hence many researchers continue to focus on the obstacles and opportunities that cloud brokering presents today. Indeed, several works like the STRATOS broker and the work presented by (56) do not define a common description model or template to use for SLA specification. This shortcoming makes the communication between the different cloud actors difficult and consequently it could lead the proposed solutions to miss their initial target. To overcome this problem, other works like SLA@SOI, Contrail and Optimis try to offer frameworks that can be integrated in cloud providers, but are usually heavy to maintain and hard to customize (67). For example, the SLA@SOI project propose an automated SLA management solution that could be integrated in many existing solutions. However, this solution can not be fully adopted in an infrastructure composed of unreliable resources such as the ones targeted by the C@H project presented in (67). Moreover, the Contrail project imposes strong constraints to providers by forcing them to integrate brokering mechanisms which hinders their adoption (66)

Besides the SLA management and negotiation issues, the brokers focused also on the service selection. They proposed decision making methods to select the most suitable service provider based on the users SLA requirements. However, some works such as Optimis and Contrail focused on specific parameters for taking their decisions which may not cover all users expectations. This lack could lead to a wrong choice of the service provider. Moreover, most brokers considers the performance criteria in their selection and neglect non functional criteria like data integrity , trust and security. Those criteria are essential in the provider's selection and they need to be formalised to be taken into account in the selection process.

We should also note that several brokers are based on a federation, such as mOSAIC and OPTIMIS. Therefore, they implement a brokering system that gather resources from different cloud providers and offers them in a custom way to their users (67).

However, the federation is not yet widely adopted by actual cloud providers especially public clouds, so it is hard to use proposed architecture in this context.

Chapter 3

Cloud Broker Architecture for negotiating semantic SLA contracts

3.1 Problem Statement

Cloud Computing has become a popular paradigm for IT services delivery allowing scalability and elasticity. However, due to the existence of various infrastructure providers (private, public and hybrid providers), the choice of the best provider's offering that fits user requirements is a complicated task. Indeed, cloud customers face the problem of heterogeneity of services and offers. This heterogeneity appears in:

- Different terminologies to express the same offer or requirement. For instance, the processing capacity of a cloud virtual machine could be called CPU, capacity, processing capacity ... as illustrated by figure 3.1. So, customers might miss the best service or even do not find the service that fits their requirements. Moreover, providers are likely to miss customers because of differences in terminology and negatively affect their benefits.

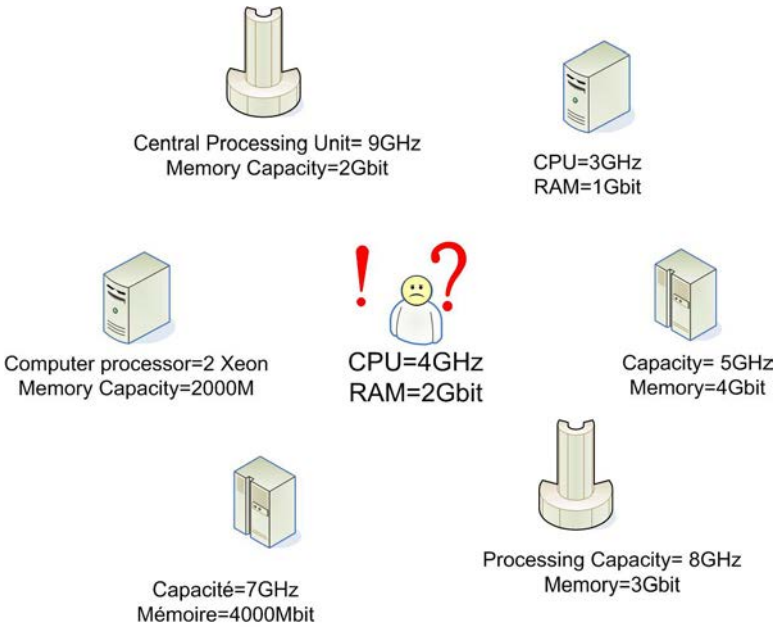

Figure 3.1: Terminology heterogeneity -

- Different formats or even different languages to express the request. Indeed, some providers like Amazon (68) and Windows Azure (69) propose predefined bundles and expect users to formulate requests as a set of required bundles. Whereas, some private clouds just ask users to fix the technical characteristics needed within the

allocated resources such as RAM, storage and data transfer rates.
This means that whenever the user requests a provider, he should comply with the provider's nomenclature which creates a dependence between customers and providers. Besides, in case of federated services when the customer needs to ask several services from various providers, he should adapt his request to provider's templates and formats which generates additional cost and delay. This problem is illustrated by figure 3.2.

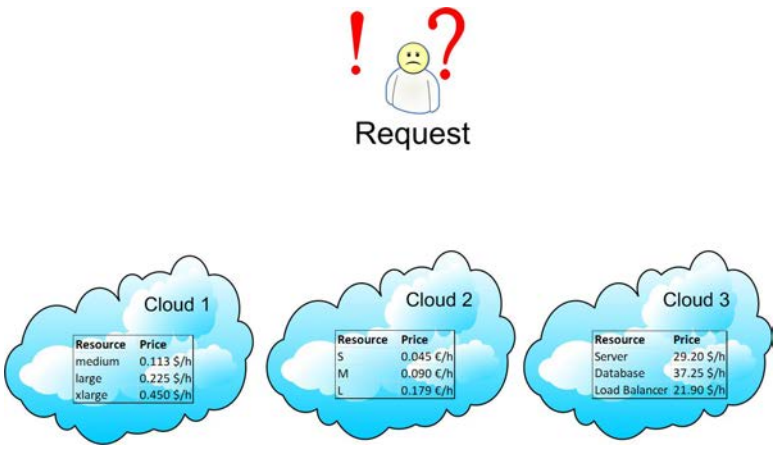

Figure 3.2: Request formulation heterogeneity -

- Different ways to express the quality of services provided. QoS parameters can be different depending on the type of the service (70) i.e. the same QoS parameters may have different meanings. For example, the QoS parameter "Delivery" expresses the percentage of service delivered without packet loss. It represents the packet delay. This parameter could be called "Loss" or "Delay" depending on the provider's choice. The latency parameter defines the waiting time per service transaction. It usually includes the transport time and querying delay. Latency could be called also "Delay". So, the QoS parameter "Delay" definition varies depending on the utilization context. Therefore, there is a risk of making the wrong choice because of a misunderstanding of the QoS parameter. This conflict is due to the absence of standard defining the quality of services which causes an ambiguity and misunderstanding between providers and consumers leading to customers not satisfied.

Previous work tried to tackle some of these issues using Service Level Agreement (SLA) contracts. An SLA contract is a formal contract used to guarantee that the service quality expected by the user will be provided. Up to now, the SLA definition is not yet standardized. Hence, cloud actors use standards like WS-Agreement (34) and WSLA (31) to express cloud SLA contracts. These formalisms are general xml-based languages largely used for SLA contracts. They are not intended to be specific to any type of market which makes them difficult to use for computing services (71). Moreover, their xml syntax offers only syntactic interoperability which is poor and not extensible (72) and can not solve the heterogeneity problem.

In addition, SLA contracts established are based on definitions proposed by providers without enabling customers with sufficient negotiation opportunity (73). It was emphasised in the technical report of the Cloud Standards Customer Council that SLA contracts offered by cloud providers are immature (74). Current SLA contracts express only application-specific performance and do not contain provisioning restrictions, for example which location to use for data storage (75). They do not offer neither performance guarantees for cloud services and leave SLA violation detection to the customer (3). Unfortunately, the customer inconvenience is not limited to the SLA contract establishment. Actually, the availability of a multitude of cloud services from various providers is a great opportunity that causes nevertheless difficulty to end users in selecting services and providers. Customers should be able to trade-off multiple QoS parameters to select the provider that best fits the desired QoS requirements as illustrated by figure 3.3. This is not an obvious practice due to the changes in end-users requirements, the dynamic fluctuations of infrastructure properties and the problem of heterogeneity mentioned above.

In this work, we are interested in ensuring:

1. Interoperability: by introducing semantic annotations in the QoS negotiation process

2. Optimum user satisfaction: find him the best trade-off between QoS parameters based on his own preferences.

3. Sustainability of user satisfaction: by providing him tools to supervise easily the compliance with his SLA requirements.

Figure 3.3: Trade-off between QoS parameters -

To achieve the stated objectives we:

1. Define a cloud specific ontology describing the semantic cloud SLA contract;
2. Propose a multi-criteria method to select the provider's offer, meeting best the user requirements and expected quality of service;
3. Propose policy to express inference rules and detect violations in the SLA contract.

These different contributions are structured in a global architecture called cloud broker architecture helping users to find adequate resources and services. Within this architecture, the SLA contract is expressed using an ontology and managed using policies and rules. With the use of ontology, an SLA contract can be richly and semantically formulated. Rules express policies governing SLA contract automation.

3.2 CBA: A Cloud Broker Architecture

Cloud computing is a new paradigm that plays a major role in the future Internet of Services by offering on-demand access to shared resources and services. But, it still faces certain terminological ambiguity. Grozev et al (75) proposed a taxonomy and a classification of inter-cloud application brokering mechanisms. In this work, we propose

an SLA-based brokering approach defined by (75) as follows : "*application developers specify the brokering requirements in an SLA in the form of constraints and objectives. The cloud provider or the Inter-Cloud service acting on behalf of the client decides on brokering approach honouring the specified SLA.*"

We introduce an SLA-based cloud broker to assist cloud consumers in managing the provisioning of cloud services. This single interface is designed to treat different requests with different requirements depending on the user preferences. Indeed, some users favour allocating resource having minimal cost, others, prefer high access to data storage resources whereas some resource demanding applications require the provisioning of scalable and elastic resources. To handle the diversity of requests, the broker analyses the demand, addresses providers to find resources and services satisfying the QoS needed and selects the best proposal. It helps also detect failures in cloud infrastructure and it alerts users of SLA contract violations. Our proposed solution is easily used by individual users. It is also time and energy saving for enterprises. It could be adopted also by providers to control their own infrastructure and manage federations. Figure 3.4 illustrates the architecture of our cloud broker.

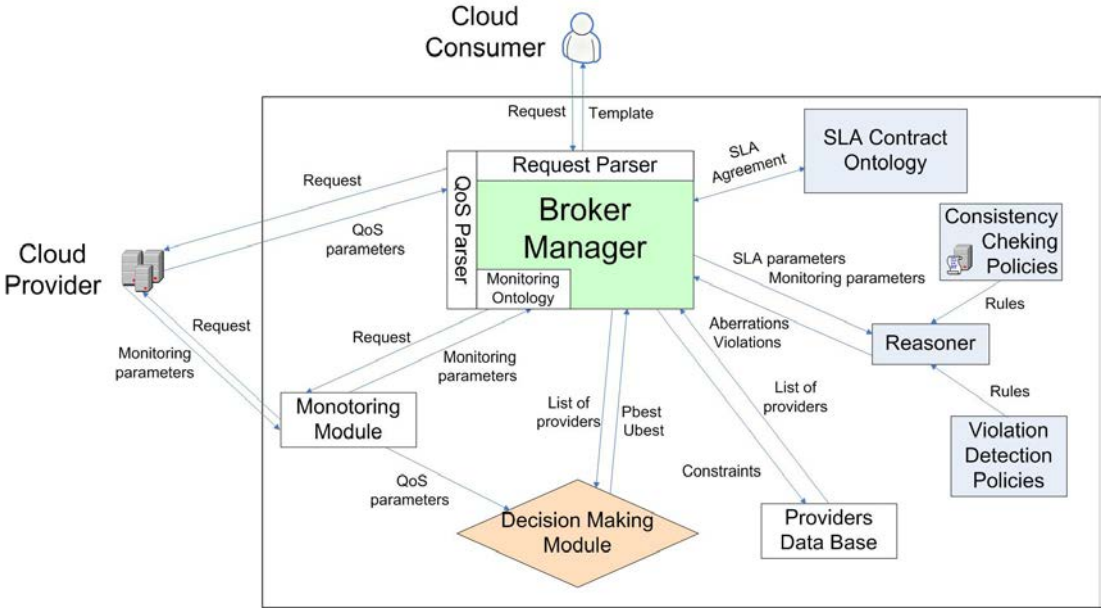

Figure 3.4: Cloud Broker Architecture -

The modules of the CBA architecture are:

- Broker Manager: It is the main module of the broker architecture that orchestrates the broker operation. It is equipped with:
  - a Request Parser Module that provides the users with a template to integrate their requests, then parses users requests to ontology;
  - and a QoS Parser Module allowing to interact with providers. It provides them with received user requests and parses the QoS parameters proposed by the providers, to answer the user requests, to ontology.

  The broker manager receives information about the user request and its preferences, find the best-fit provider by interacting with the broker's decision making module and establish an ontological contract with the selected provider. To ensure that this SLA contract is respected, the broker manager interacts with the reasoner module to detect violations, then it reacts by imposing penalties and migrating user applications to another provider if necessary.

- SLA Contract Ontology: describes parameters to be introduced by both customers and providers to establish and negotiate a coherent cloud SLA contract. It is a semantic module ensuring interoperability between heterogeneous cloud actors.

- Reasoner module: is a program used to derive new facts from the existing ontologies. It is invoked to:
  - verify the consistency of the consumer's request based on the consistency checking rules provided by the Consistency Checking Policies Module.
  - detect violations: it checks that the real-time monitored QoS parameters do not violate the agreed SLA contract based on the Violation Detection Policies. Otherwise, it alerts the broker manager of the occurred violations.

- Consistency Checking Policies module: contains rules to verify the consistency of a request.

- Violation Detection Policies module: contains rules to check if an SLA parameter is violated and determine subsequent penalties.

- Decision Making Module: this module evaluates provider's performances against the heterogeneous SLA criteria defined in the cloud ontology by assigning utilities of each provider. It adopts a multi-criteria decision making method for providers ranking associated with utility function descriptions to determine user satisfaction degree.

- Providers knowledge Base: Contains information about collaborative providers such as the geographic locations of their data center, their replication method or their recovery strategy.

- Monitoring module: SLA contracts negotiated by the broker includes quality of service requirements and penalties in case of QoS violations. However, to detect that the QoS requirement are not met any-more, services hosted by the providers must be monitored. This module measures in real time the QoS parameters of these services. The monitored parameters allow to detect the violated QoS performance agreed by the consumer and the provider. Several techniques could be used for monitoring, such as traditional server monitoring services that can be used to likewise monitor cloud services, vendor specific monitoring services like Hyperic (76), CloudHarmony (77), Monitis (78), Nimsoft (79), Amazon CloudWatch (80) ... or even third party independent cloud monitoring services like Cloudstatus (81) and cloudkick (82). We suppose that this module holds a database of monitored QoS that helps the Decision Making Module to select a provider for the user requests.

This architecture ensures three main features:

1. **Establish a coherent SLA contract**:
   The life-cycle of an SLA contract starts when the broker receives a new request. Right away, it creates a new SLA Ontology Contract based on a generic contract model already stored, and provides the user with a clear template to express his requirements and preferences. The broker asks the user about his resource needs as well as his service level objectives. However, broker's customers are not necessarily experimented people able to formulate coherent requests. So, the broker is equipped with a Consistency checking Policies module enumerating good practices to formulate coherent requests more likely to be answered. To verify the

consistency of the user request, the broker manager solicits the Reasoner module that is able to understand consistency rules to detect eventual aberrations. In that case, the broker manager alerts the customer about the detected aberration and asks him to reformulate his request if he wants. This feature is illustrated by the sequence diagram 3.5.

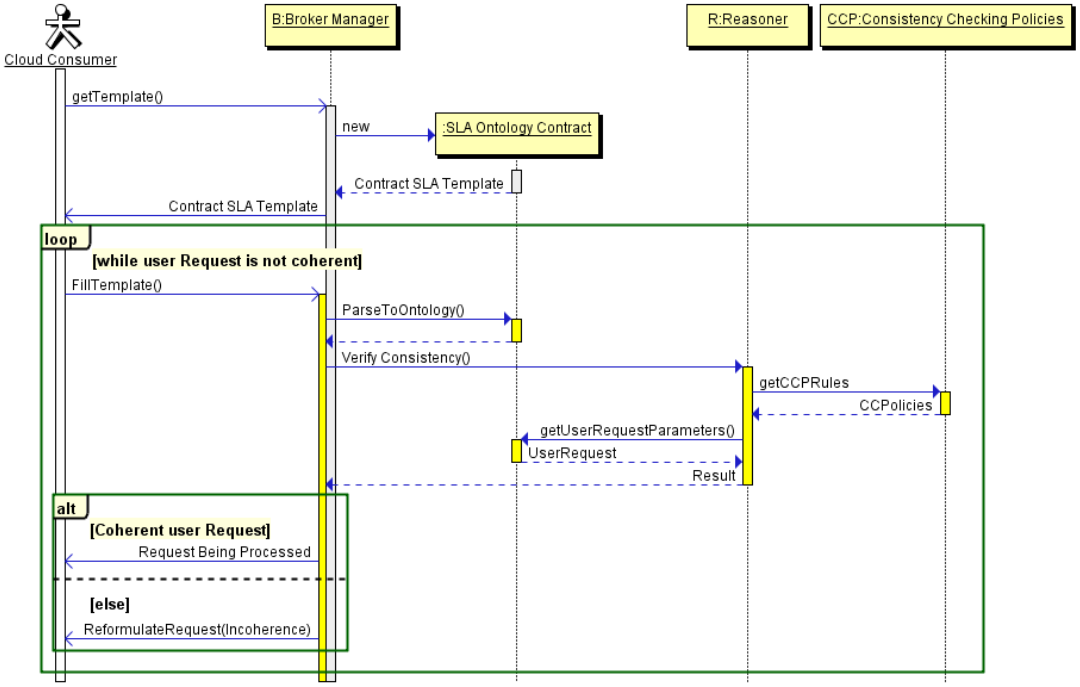

Figure 3.5: Coherent SLA Establishment -

2. **Select the best provider and establish the agreement between the customer and the provider**:
   The broker's main goal is to find the best provider to the user request and establish an SLA contract between the consumer and the selected provider. This process goes through several steps:

   (a) the broker considers the user constraints: it examines its knowledge base to select the providers capable of satisfying the user constraints.

   (b) the broker's decision making module finds the best suited provider: first of all the decision Making Module predicts the QoS that the provider can supply based on the database monitoring information. This database holds the QoS

values measured by the monitoring module responsible for tracking the user requests and measuring real time QoS values. Based on the predicted QoS values, the DMM calculates the utility of each provider. Then, it sorts the providers utilities and determines the best provider to the user request.

(c) Finally, the broker manager contacts this provider and establish the SLA contract between the consumer and the provider.

Those steps are illustrated by figure 3.6

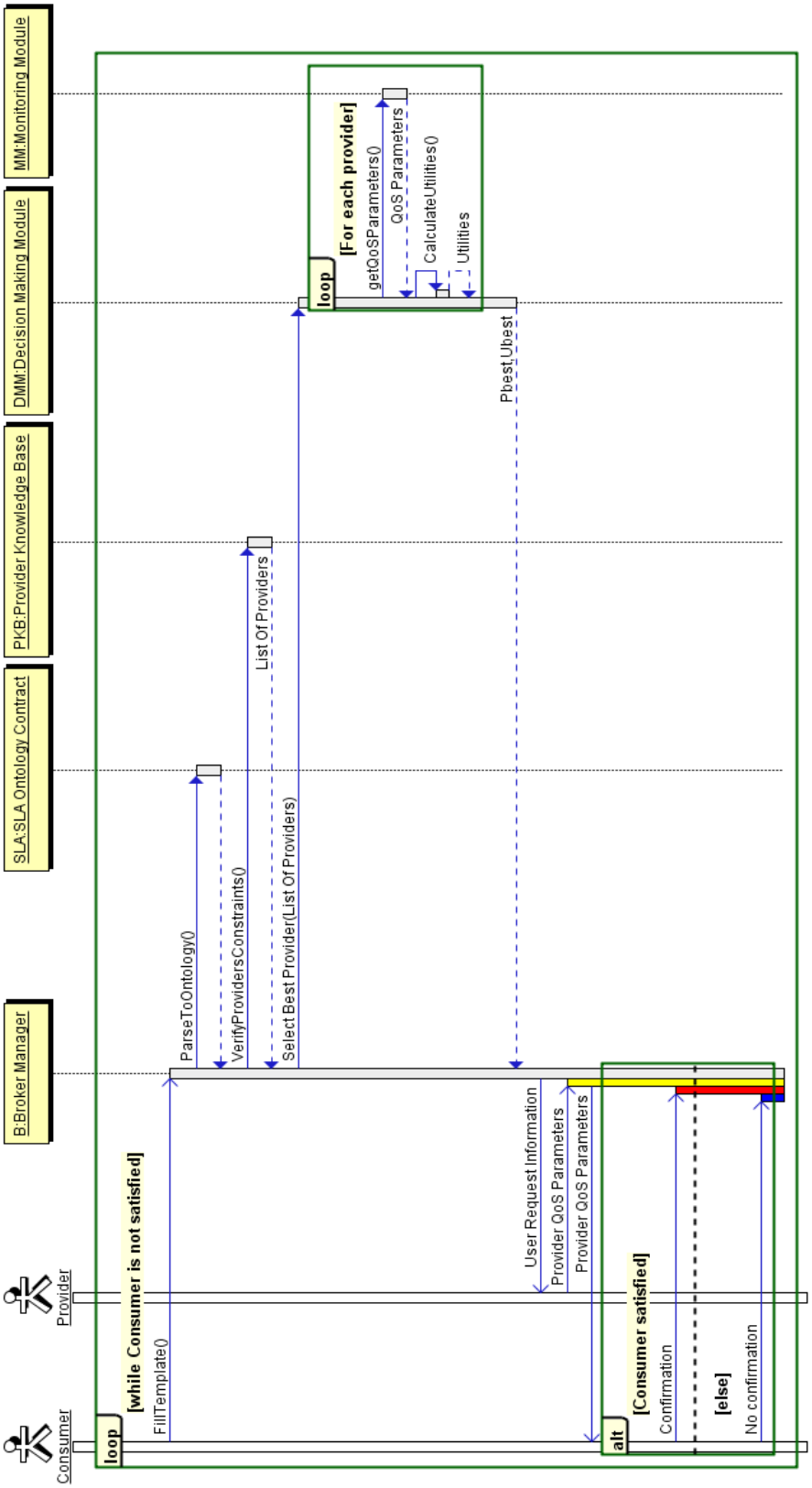

Figure 3.6: SLA Negotiation Process -

3. **Detect violations in the SLA contract**:

   The broker's role is not limited to the establishment of the SLA contracts between providers and consumers. It also ensures that this contract is respected. Indeed, it periodically checks the compliance of the provided quality of service with the service level objectives (SLOs) agreed between the provider and the consumer. The violation detection scenario is illustrated by the sequence diagram in figure 3.7.

   The broker manager receives periodically the QoS values from the monitoring module. It communicates this information to the Reasoner that detects the violated SLOs and alerts the broker manager of the penalties agreed. So, this latter looks after imposing those penalties to the provider. Moreover, if the provider does not satisfy any-more the user request, the broker resorts again to the DMM to select another provider and migrate user applications to the new provider.

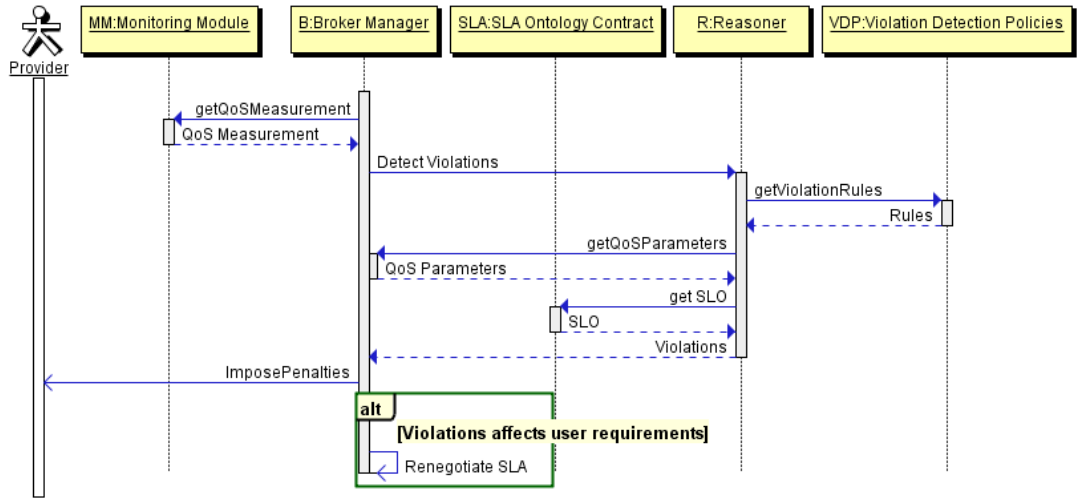

Figure 3.7: SLA Violation Detection Process -

In our proposed architecture, the feature of establishment of a coherent SLA contract as well as the detection of violations is assured thanks to the policies defined by the broker manager and interpreted by the reasoner. In the next section, we will detail how the broker implements these policies.

3.3 Management Policies Implementation

The increasing complexity of cloud architectures has made manual management not adequate. Infrastructures need to be provided with exceptional capabilities to enable it automatically. In this work, we propose to define policies to automate the management of the SLA contract. The cloud Broker architecture proposed in the previous section allows the establishment of SLA contracts between consumers and IaaS, PaaS and SaaS providers. For each type of contract, adequate policies should be specified to handle the coherence of the user request and to define sensible penalties to manage violations. This could be done based on the specific domain ontology administrating the negotiation of the SLA contract. In fact, ontologies are responsible for defining the decisional domain taxonomy, i.e., the concepts and the relations between them. They also provide a unified way of message exchange between different contributors of the decisions. Hence, by the use of ontologies, we can define a shared referential to communicate between different entities interacting in the contract establishment. According to the concepts defined in the ontology, we can define management policies namely the request consistency policies and violations detection policies. All management policies should be in accordance with the ontological concepts. To configure our broker architecture, the proposed architecture is provided with an interface to help a management agent to write management policies. This interface is illustrated in Figure 3.8.

For manipulating ontologies and expressing policies, we used the Jena API (83). It is a Java based API for handling OWL (84, 85) and RDFS (86, 87) ontologies and it provides a rule-based inference engine for reasoning over RDF (86) and OWL (84, 85). Our interface supports the broker managers in expressing policies as Jena rules. This flexibility facilitates and encourages stakeholders and providers to adopt our brokering solution since there is no need to learn a new policy language. Managers need only load the domain ontology they are going to use. The system charges automatically subjects, predicates and objects separately to help them to create triple patterns. Managers could easily include Jena built-ins in forward or backward chaining rule depending on the rule reasoned configuration. Built-ins express conditions to be fulfilled in the antecedent part of a policy as well as relations between concepts to be considered as post-conditions once the policy is executed. In this way, managers establish swiftly consistency checking rules and violation detection rules based simply on a domain ontology.

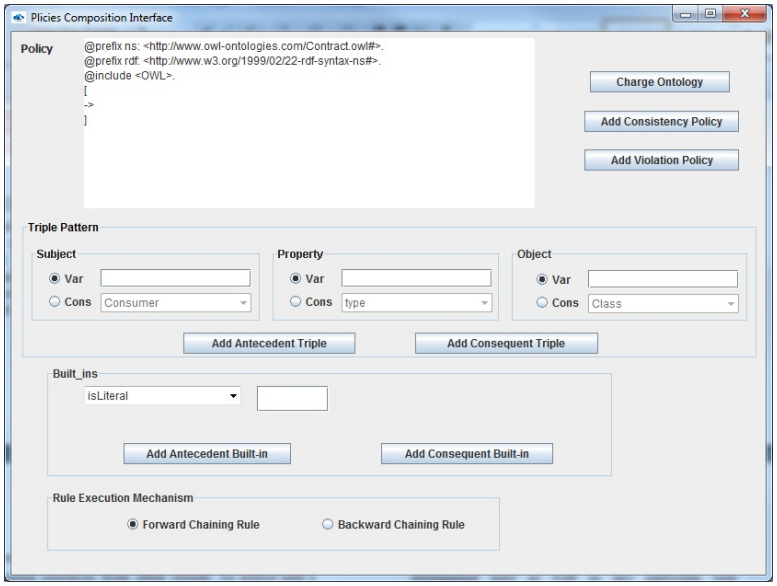

Figure 3.8: Management interface to create policies -

3.3.1 Consistency Checking Policies

Consistency checking policies are expressed as normative rules. Their main objective is to warranty that user requests are semantically coherent. We use integrity constraints to ensure consistency and coherence of the knowledge bases provided by customers. Indeed, the domain ontology gives a template to be followed while specifying requests or expressing SLA criteria but does not necessarily emphasize coherence conditions to be fulfilled throughout the system's life cycle. Therefore, the consistency checking policies prevent users in formulating irrational queries so queries are more likely to be answered. For example, if the user needs IaaS virtual machines with high CPU capacity and rapid memory, he is not allowed to fix a very low cost threshold. Otherwise, the request can not be fulfilled. In order to avoid such aberrations, the broker manager includes the following policy:

if (Compute-cores> x and Compute-memory > y and cost < z)
then AlertUser ("The requested resources should be more expensive than the fixed cost threshold")

This policy could be formulated as illustrated by figure 3.9.

This module includes also functional dependencies between QoS metrics. For example, the storage and the network QoS metrics are strongly coupled. Then, when

```
@prefix ns: <http://www.owl-ontologies.com/IaaS_SLA_Contract.owl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

@include <OWL>.

[PriceConvenience:
(?Compure_Resource ns:compute_cores ?a)
(?Compure_Resource ns:compute_memory ?b)
(?Functional_Thresholds ns:maxCost ?c)
greaterThan(?a, x)
greaterThan(?b, y)
lessThan(?c, z)
->
(ns:PriceConvenience ns:alertUser InsufficientCostThreshold)
]
```

Figure 3.9: Consistency Checking Policies Example -

querying for a rapid access to storage space, we can not ask for a very small read time and write time all over with a minimum bandwidth capacity. The policies of this module are adjusted by the broker manager in accordance with its environment and context knowledge.

3.3.2 Violation Detection Policies

Periodically, the broker manager receives the monitored QoS measures that it communicates to the reasoner to verify if the QoS variations measured could lead to violations in the SLA contract. To detect violations, the reasoner infers the policies stored in the Violation Detection Policies engine. In our architecture, the definition of penalties are dissociated from the SLA contract establishment. The broker manager negotiates in advance the penalties to be imposed in case of violation with the providers. Then, the broker manager introduces the violation detection policies as rules in the violation detection file. Since the broker holds the user preferences as well as the proposed agreement by the provider, two types of treatments are possible after detection a violation:

- If the violation affects the SLA contract but still satisfies user needs, only penalties are imposed to the provider.

- If the violation exceeds or falls below the thresholds fixed by the consumers, the broker concludes that the provider does not satisfy anymore the user request. It imposes penalties to the provider and treats the user request again to find another provider that can meet the request requirements.

An example of detection of response time violations is illustrated by figures 3.19 and 3.11. The first figure illustrates the detection of violations that do not affect the user requirements whereas the second illustrates the detection of severe violations that require the decision making module intervention.

```
@prefix ns: <http://www.owl-ontologies.com/IaaS_SLA_Contract.owl#>.
@prefix nf: <http://www.owl-ontologies.com/QoS.owl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

@include <OWL>.

[ResponseTimeViolation1:
(?Functional_SLO ns:SLArt ?x)
(?Functional_Thresholds ns:maxResponseTime ?y)
(?Functional_QoS nf:QoSLatency ?z)
(?net ns:hasMaxLatency ?lat)

greaterThan(?z,?x)
lessThan(?z,?y)
->
(nf:ResponseTimeViolation ns:isViolated "True"xs:boolean)
(nf:ResponseTimeViolation ns:hasPenalty "5"^^xs:integer)
(nf:ResponseTimeViolation ns:hasPenaltyMeasurementUnit "USD")
]
```

Figure 3.10: Response Time Violation Detection Policies Example 1 -

```
@prefix ns: <http://www.owl-ontologies.com/IaaS_SLA_Contract.owl#>.
@prefix nf: <http://www.owl-ontologies.com/QoS.owl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

@include <OWL>.

[ResponseTimeViolation2:
(?Functional_SLO ns:SLArt ?x)
(?Functional_Thresholds ns:maxResponseTime ?y)
(?Functional_QoS nf:QoSLatency ?z)
(?net ns:hasMaxLatency ?lat)

greaterThan(?z,?x)
greaterThan(?z,?y)
->
(nf:ResponseTimeViolation ns:isViolated "True"xs:boolean)
(nf:ResponseTimeViolation ns:hasPenalty "5"^^xs:integer)
(nf:ResponseTimeViolation ns:hasPenaltyMeasurementUnit "USD")
(nf:ResponseTimeViolation ns:exceedsThreshold "True"xs:boolean)
]
```

Figure 3.11: Response Time Violation Detection Policies Example 2 -

Our cloud broker is now well configured and ready to start the negotiation of the SLA agreements. In the following, we will discuss the cloud SLA contract specification.

3.4 Cloud SLA Contract specification

There are a multitude of specifications and formalisms for defining SLA contracts. However, they are generic models, widely used in various domains, not necessarily

adapted to the cloud. Nowadays, there is no standardized manner to establish cloud SLA contracts. Cloud actors usually use xml standards like WS-Agreement and Web Service Level Agreement, which are hardly adapted to the cloud. Moreover, the lack of a standard taxonomy for the cloud domain causes many heterogeneity problems. To handle this kind of issues, we introduce semantic annotations in the cloud SLA contracts. Ontology is a flexible semantic tool, not exploited for SLA specification so far. In this work, we build an ontology for cloud SLA specification formalizing the SLA contract between consumers and providers. The main concepts of this ontology are the two parties of an agreement which are the consumer and the service provider as illustrated by Figure 3.12:

- Consumer: authenticated and secured via a login and a password. It is the service requester which could be a human person or an external entity such as the cloud platform layer or a cloud provider requesting resources from other clouds. To notice user's faithfulness, we define an incremental attribute in the ontological contract to calculate the number of SLA contracts endorsed by each user through the broker;

- Provider: the provider represents the second partner in the SLA contract. It is associated to the service type it provides. It is essential to specify the provided service type especially in a federation scenario to be able to determine which provider supplies which service. To have an idea about the provider performance, we calculate the number of SLA contracts it established;

- Duration: It represents the engagement duration. Each SLA contract has a start date and an end date.

The consumer initiates the negotiation of the SLA contract by introducing his requirements gathering his request, his specific constraints if he has any, the QoS he needs and his engagement duration as illustrated by figure 3.12. To answer this request, the provider checks his available resources at the specified duration and provide the customer with the QoS he is able to guarantee.

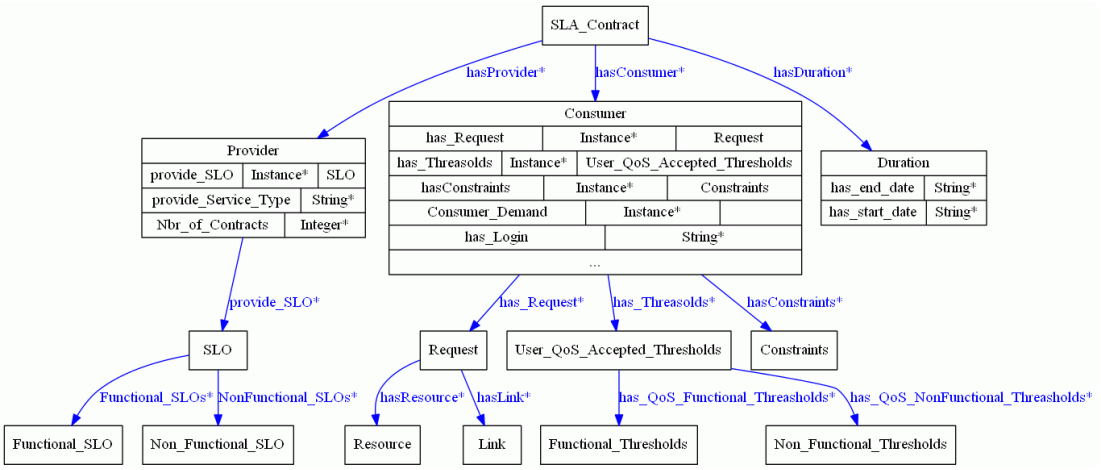

Figure 3.12: Cloud SLA Contract Ontology -

To hide the heterogeneity between different concepts and relations, we introduce semantic annotations in the proposed ontology. For instance we use "same as" relation to identify that two individuals are the same (SLO=Service Level Objectives) and the "equivalent property" relation to indicate that two properties are equivalent (compute-core=CPU).

The proposed ontological contract is implemented and tested for IaaS SLA contracts.

3.4.1 IaaS Cloud SLA Contract specification

The cloud SLA contract is specified for IaaS requests as illustrated by figure 3.13. IaaS consumer's requirements includes:

- Consumer request: to request for Infrastructure As A Service (IaaS) resources, the user should specify his/her requirements in terms of virtual resources. Our request is conform to the IaaS OCCI specification (88). It can include three types of resources: Network, Compute and Storage. The attributes of each type of resource is defined by the OCCI specification. In order to create entities like virtual data centres or virtual clusters, we should define links between the different resources. Those links are defined as instances of the class Link. There are two types of links: the storage link connecting a compute resource to a storage resource and the Network Interfaces connecting the compute resources to Network resources;

- Constraints: Sometimes, consumers have crucial constraints affecting the provider's choice but that can not be expressed in the request to the provider. The geographic location of the stored date, for instance, is not explicitly determined in the storage resource request, so the provider is free to store resources worldwide. Thanks to the constraints field of our ontology, IaaS consumers can introduce their preferences regarding the geographic location of their stored resources, their recovery strategy (dedicated or shares recovery) and the replication type they need (static or dynamic replication).

- User QoS Accepted Thresholds: SLA criteria specifies the service level objectives of the SLA agreement. SLA metrics can be classified by different categories. Table 3.1 enumerates negotiated SLA categories and the metrics measured in each one. Negotiated SLA criteria are classified in two main categories: functional properties and non functional properties. Functional properties define properties that directly influence quality of service. They are classified in four categories: compute SLA, Network SLA, Storage SLA and cost. However, non functional properties are rarely explicitly described in a user request but can be used as indicators of quality. We grouped scalability, elasticity and disaster recovery in a reliability family defining the ability of the system to react to emergency situations. Moreover, we considered trust as an important non functional evaluation criterion since it defines the trustworthiness of a service provider. User QoS Accepted Thresholds indicate accepted thresholds for the user for the mentioned criteria.

When receiving the customer's request, the provider introduces his proposed SLA criteria. The values proposed by the provider must satisfy the user request and they represent the service level objectives (SLO) that it must maintain.

For cloud SLA specification, we build an ontology enabling semantic interoperability. Based on this ontology, the broker builds clear and complete templates that it communicates to the consumers for expressing their requests and their preferences. Then, the broker has the responsibility of finding them the best provider. Thereby, consumers get rid of the burden of communicating with heterogeneous providers to select the best one. However, how could the broker communicate with different providers utilizing heterogeneous SLA languages? Should they imperatively adopt the ontological concepts? The

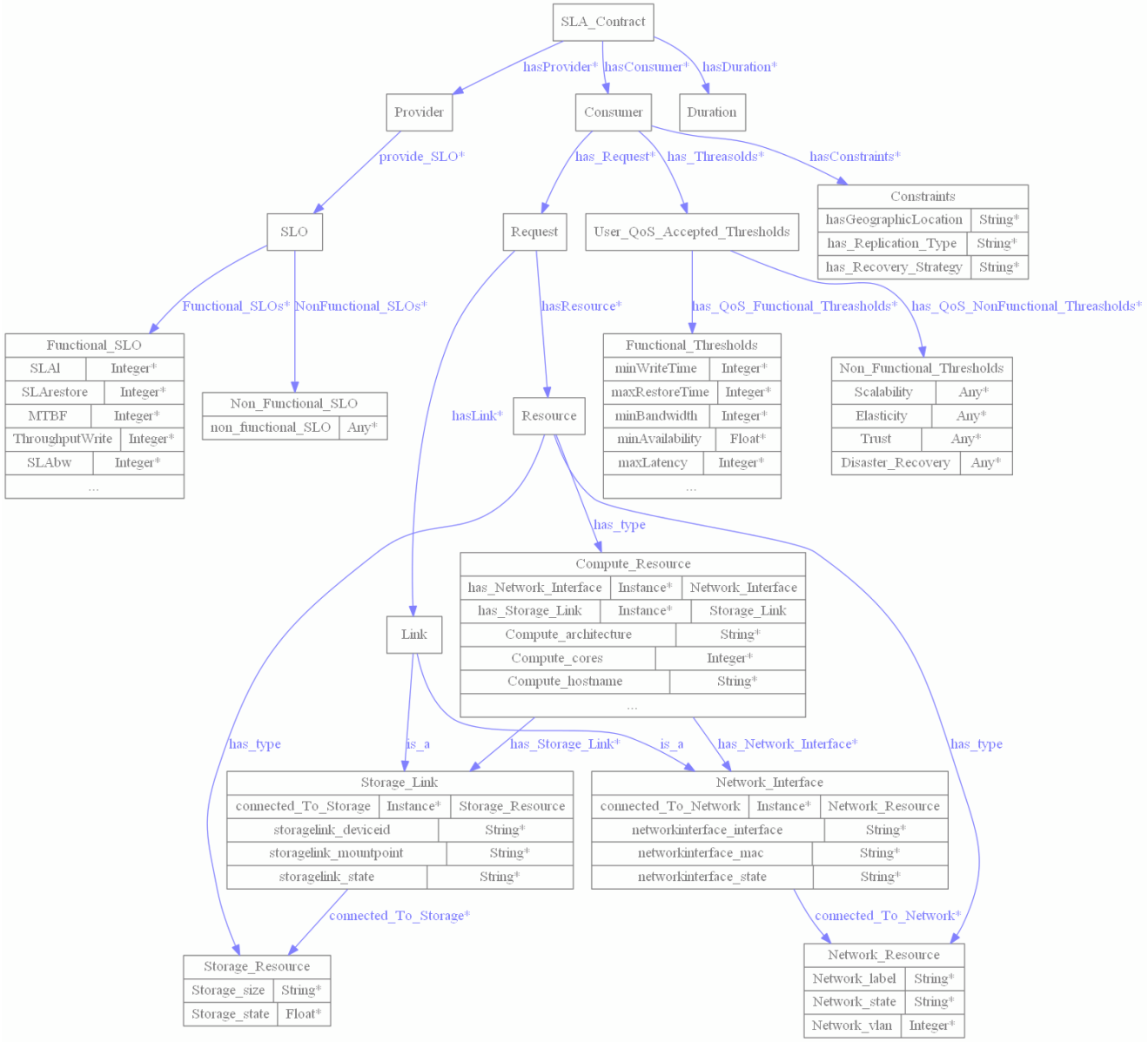

Figure 3.13: IaaS SLA Contract Ontology -

Table 3.1: Negotiated SLA criteria classification.

	Category	Measurable Metrics	
Functional Properties	Compute SLA	Response Time	
		Availability	
	Network SLA	Latency	
		Bandwidth	
	Storage SLA	Performance	Read throughput
			Write throughput
		Recovery(restoreTime)	
	Cost		
Non Functional Properties	Reliability	Scalability	
		Elasticity	
		Disaster Recovery	
	Trust		

cloud SLA specification is not yet standardized. There are several xml-based languages like WSLA and WS-Agreement to make SLA agreements. To allow providers to communicate with the broker their SLA formalisms should be mapped to ontology. Since WS-Agreement is widely used by cloud SLA actors, we will explain in the next section how WS-Agreement could be mapped to ontology.

3.5 Ontology Mapping to WS-Agreement

For cloud SLA specification we build an ontology enabling semantic interoperability. This ontology introduces a specification of cloud SLA negotiation. It contains all elements of an SLA Agreement. Thus a cloud SLA negotiation contract expressed in another language could be easily mapped to ontology and vice versa. To hide the heterogeneity and ensure interoperability in the cloud, the broker resorts to mapping techniques as illustrated by figure 3.14.

This figure shows that providers using WSLA for instance can communicate with providers using another SLA language like WS-Agreement by mapping WSLA to ontology, then mapping ontology to WS-Agreement. Thereby, our ontology is enriched by the concepts of other SLA languages.

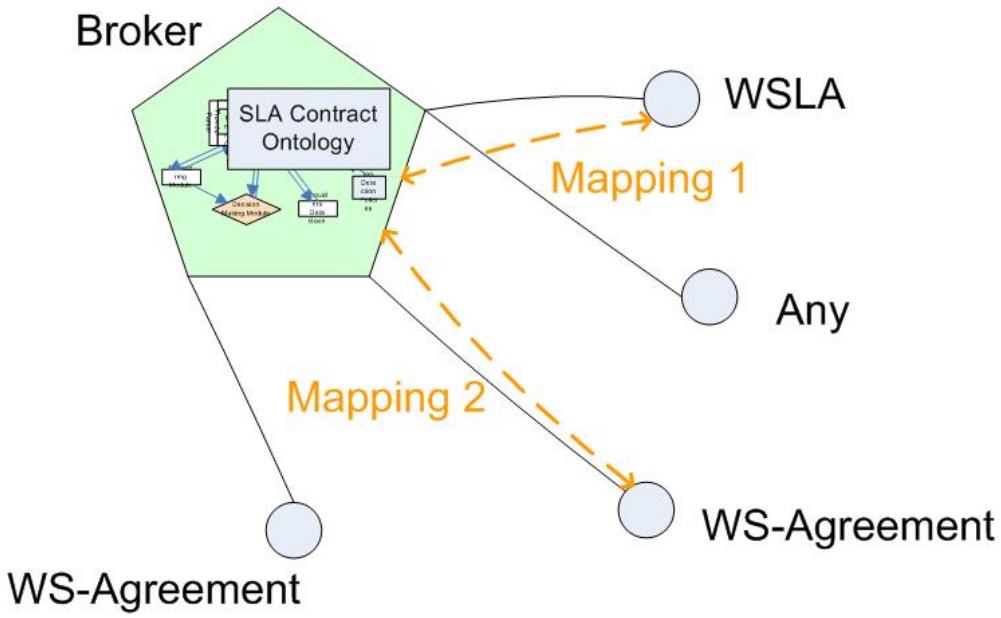

Figure 3.14: SLA interoperability via the broker -

WS-Agreement is an SLA language widely used for cloud SLA specifications. It is a normative language to formulate Service Level Agreements and a basic protocol to expose service-level descriptions, validate service-level requests, and come to an agreement. This Java framework, proposed by the OGF WS-Agreement standard, defines a tool to create and manage service level agreements in distributed systems. The structure of an agreement template is summarized by the figure 3.15.

The objective of the WS-Agreement specification is to define a general language for creating agreements based on templates. However, since WS-Agreement is not intended to be specific to any type of market, it is very difficult to be used for computing services (71).

In the following, we will define the mapping term by term of WS-Agreement to ontology. The mapping notation used is inspired from (89) and defined by table 3.2.

The first part of the agreement template is the context containing information about the agreement initiator, the agreement responder and the expiration time. The mapping of context information to ontology is defined as follows.

mp1=(Consumer, /Template/Context/AgreementInitiator)

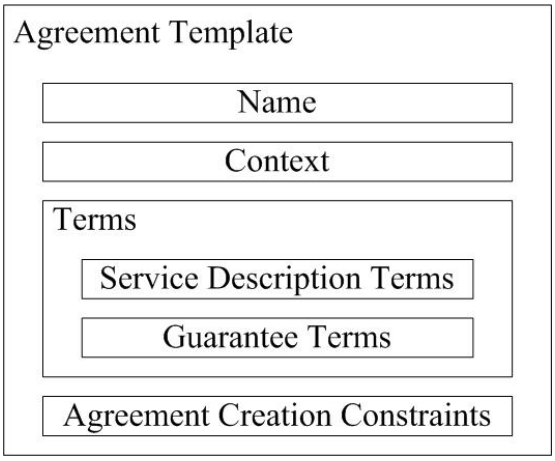

Figure 3.15: Structure of WS-Agreement template -

Table 3.2: Mapping Notation.

Mappings	**Notation**
Class	(OWL Class URI, XPath expression)
Datatype Property	(OWL Datatype Property URI, Domain Class Mapping, XPath Expression)
Object Property	(OWL Object Property URI, Domain Class Mapping, Range Class Mapping)
Datatype Property of class Instance	(OWL Datatype Property URI, RDFType:Instanceof Domain Class Mapping, XPath Expression)
Object Value of Datatype Property of class Instance	(OWL Datatype Property URI, RDFType:Instanceof Domain Class Mapping, Object Range, XPath Expression)

mp2=(Provider, /Template/Context/AgreementResponder)
mp3=(hasEndDate, Duration, /Template/Context/ExpirationTime)

The agreement initiator is mapped to the concept "Consumer" since the brokering process starts by a consumer's request which could be a provider, the agreement responder is mapped to the concept "Provider " which is the cloud service responder and the expiration time is mapped to the the datatype property "hasEndDuration" of the concept "Duration".

In the "terms" section WS-Agreement defines the service terms and the guarantee terms. Service Terms define services that are traded. In the cloud context, this section describes user request:

mp4=(Request, /Template/Terms/ServiceDescriptionTerm)

Requests are about an IaaS, PaaS or SaaS service. In case of IaaS SLA contract, the request represents the tree presented by figure 3.16.

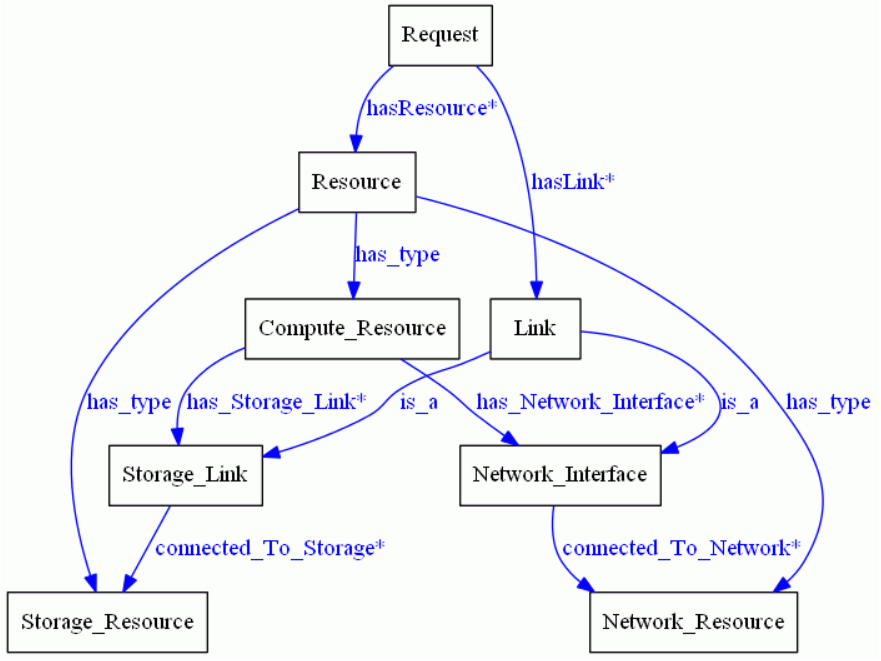

Figure 3.16: IaaS Request -

The WS-Agreement service property attribute defines properties related to each service. Each property is defined by its name, the service name it is applied to and a set of variables. Variables are defined by their name, their metric and their location which is a reference giving a scope to the concept represented by the variable. In the cloud context, SLA parameters such as response time and availability should be defined in this section. Properties are defined as instances of the classes Functional-SLO or Non-Functional-SLO. For each class, it is associated datatype properties representing the variables. For IaaS SLA requests, the SLOs representing the properties of IaaS services are defined by the figure 3.17.

For example, to map the response time variable of the property Functional-SLO , we apply the following operations:

mp5=(Functional-SLO, /Template/Terms/ServiceProperties@Name)
mp6=(SLArt, mp5,/Template/Terms/ServiceProperties/Variables

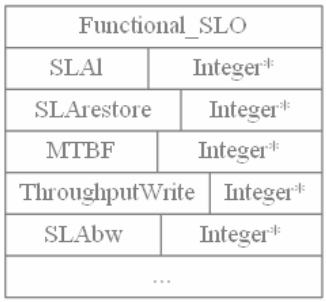

Figure 3.17: IAAS SLOs -

/Variable@Name)
mp7=(SLArt-Metric,mp5, /Template/Terms/ServiceProperties/Variables
/Variable@Metric)

"Guarantee terms" describe assurance on the service quality the provider will have to guarantee. They consist of the ServiceScope, the QualifyingCondition, the ServiceLevelObjective and the businessValueList sections. The serviceScope determines the service the guarantee are applied to. In our ontology, the guarantees are directly related to the user request. The QualifyingCondition defines the preconditions to be satisfied before the guarantee starts to apply. In the cloud Context, SLA have to be satisfied periodically throughout the service life-cycle. ServiceLevelObjectives define the key performance indicators to be respected for the service scope. They are final SLA parameters agreed by both consumer and provider. They are proposed by the provider to respond to user request and agreed by the consumer. In the SLA contract ontology, they represent the object part of the triple (S,P,O) of the instances of the variables defined in the SLO part. An example of mapping the response time agreement of the property Functional-SLO defined previously to ServiceLevelObjectives is defined as follows:

mp8=(SLA-rt, RDFType:Instanceof mp5, /Template/Terms/GuaranteeTerms
/ServiceLevelObjective/KPITarget/KPIName)
mp9=(SLA-rt, RDFType:Instanceof mp5,5, /Template/Terms/GuaranteeTerms
/ServiceLevelObjective/KPITarget/Target)

The KPIName of this service level objective is the response time of the instance of the class Functional-SLO created in this SLA contract. The target explaining the con-

dition of client satisfaction is represented by the range value attributed to the datatype property instantiated.

"BusinessValueList" section describes business objectives associated to service objectives. It defines essentially penalties and rewards. Business objectives are defined as rules extending OWL ontology. Indeed, OWL standards allow reasoning over ontologies. Meta-reasoning rules are used for facilitating meta-reasoning on ontology in controling and knowledge engineering tasks. In cloud context, we express violation detection policies as rules to detect prospective violation and to define subsequent penalties. In WS-Agreement, ServiceLevelObjectives defines conditions to be respected and Penalty attributes express penalties to be imposed for violations of Service Level Objectives. In addition, the jena rules we used to express penalties have the following form: [rulename:(tripleto match)->(triple to add to model)]. Hence, before executing penalties, conditions to be verified are written in triple to match part and penalties are expressed by additional triples to adhere to our model. Consequently, the following example illustrating a penalty expressed in WS-Agreement

```
<wsag:GuaranteeTerm wsag:Name="TransferTimeJob1">
        <wsag:Obligated>ServiceProvider</wsag:Obligated>
        <wsag:ServiceScope>

        <wsag:ServiceName>rosettaNet:getInvoice</wsag:ServiceName>
        </wsag:ServiceScope>
        <wsag:ServiceLevelObjective>
                <wsag:predicate type="less">
                <wsag:parameter>qos:ResponseTime</wsag:parameter>
                <wsag:value>5</wsag:value>
                <wsag:unit>time:seconds</wsag:unit>
                </wsag:predicate>
        </wsag:ServiceLevelObjective>
        <wsag:BusinessValueList>
                <wsag:Penalty>
                        <wsag:AssessmentInterval>
                        <wsag:Count>1</wsag:Count>
                        </wsag:AssessmentInterval>
                        <wsag:ValueExpression>5</wsag:ValueExpression>
                        <wsag:ValueUnit>USD</wsag:ValueUnit>
                </wsag:Penalty>
        </wsag:BusinessValueList>
</wsag:GuaranteeTerm>
```

Figure 3.18: Penalty example written in WS-Agreement -

could be expressed as rules.

Finally, agreement creation constraints support agreement initiator and agreement responder to define acceptable values for service descriptions. So, each offer Item has to comply with Item Constraint element and the WS-Agreement contract is not vali-

```
@prefix ns: <http://www.owl-ontologies.com/IaaS_SLA_Contract.owl#>.
@prefix nf: <http://www.owl-ontologies.com/QoS.owl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

@include <OWL>.

[ResponseTimeViolation1:
(?Functional_SLO ns:SLArt ?x)
(?Functional_Thresholds ns:maxResponseTime ?y)
(?Functional_QoS nf:QoSLatency ?z)
(?net ns:hasMaxLatency ?lat)

greaterThan(?z,?x)
lessThan(?z,?y)
->
(nf:ResponseTimeViolation ns:isViolated "True"xs:boolean)
(nf:ResponseTimeViolation ns:hasPenalty "5"^^xs:integer)
(nf:ResponseTimeViolation ns:hasPenaltyMeasurementUnit "USD")
]
```

Figure 3.19: Penalty example as Rule -

dated until each item conforms creation constraints. Such constraints could be directly introduced in our ontology due to its semantic aspect. Indeed, quantifier restrictions, cardinality restrictions and hasValue restrictions are directly introduced in our ontology. For example, to define acceptable values for response time metric, WS-Agreement creates the constraints illustrated by figure 3.20

```
<wsag:CreationConstraints>
    <wsag:Item>
        <wsag:Location>
            //qos:ResponseTime Measurement Metric
        </wsag:Location>
        <wsag:Constraint>
            <xsd:sequence>
                <xsd:element
                    name="lowerthan"
                    minOccurs="1"
                    maxOccurs="1">
                    <xsd:simpleType>
                        <xsd:restriction base="xsd:string">
                            <xsd:enumeration value="Second"/>
                            <xsd:enumeration value="MilliSecond"/>
                            <xsd:enumeration value="Minute"/>
                        </xsd:restriction>
                    </xsd:simpleType>
                </xsd:element>
            </xsd:sequence>
        <wsag:Constraint>
    </wsag:Item>
    <!-- free form constraints -->
    <wsag:Constraint />
```

Figure 3.20: Creation constraint example written in WS-Agreement -

These restrictions could be introduced directly in the ontology by creating a functional property allowed to have only some acceptable values as illustrated by figure 3.21

In conclusion, the proposed ontology contains all elements of a WS-Agreement as

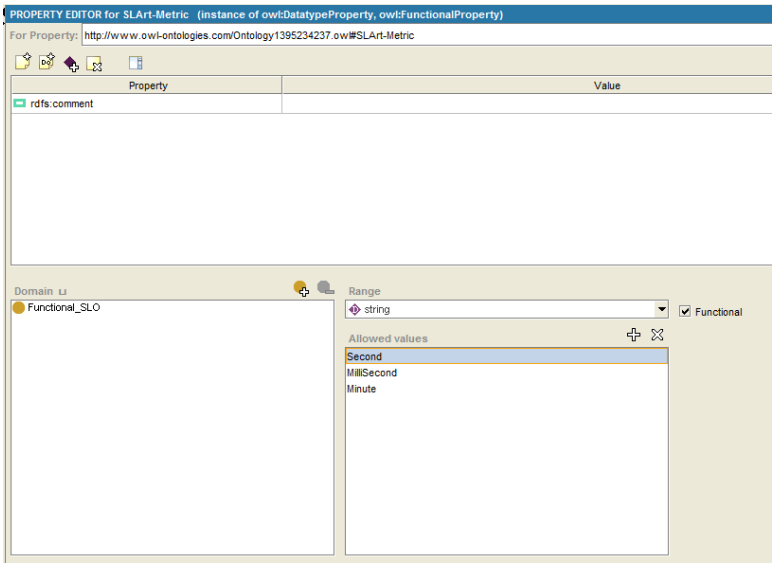

Figure 3.21: Creation constraints example created by Protégé -

illustrated by figure 3.22. Penalties are introduced separately in a rule file to enable violations detection. Moreover, the ontology is richer than xml-based SLA languages since it includes the customer's preferences (User QoS Accepted Thresholds) and introduces semantic annotations to hide heterogeneity in the cloud domain.

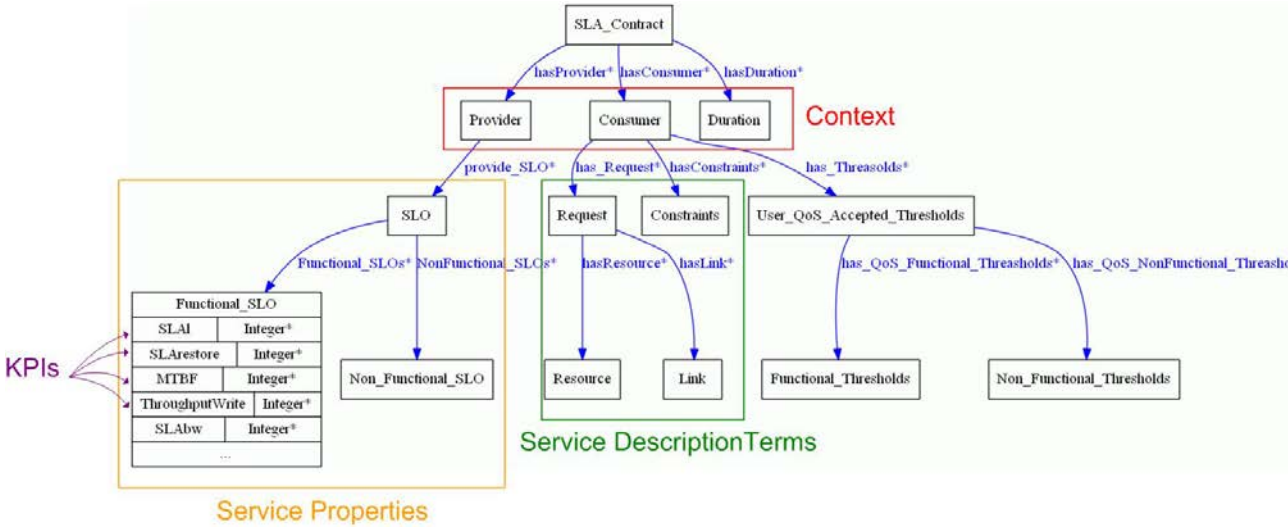

Figure 3.22: The mapping summary -

Chapter 4

Service provider's selection based on the multi-criteria method

4.1 Motivation

The emergence of Cloud computing solutions has attracted many potential customers from different natures looking for a way to reduce the costs associated with supporting their business processes. A multitude of cloud providers offer raw resources to cloud consumers enabling them to provision processing, storage, networks and other fundamental computing resources. On allocated resources, they deploy and run software including operating systems and applications.

The availability of a multitude of cloud services from various providers is a great opportunity that causes nevertheless some difficulty to end users in selecting services and providers for infrastructure, platform and software services meeting best their requirements and expected quality of service. Indeed, the adoption of cloud computing requires a detailed comparison of infrastructure alternatives, taking several QoS parameters into careful consideration. So, how could end users choose the provider's service proposing the best trade-off between multiple QoS parameters?

One possible approach, adopted in this work, is to rely on cloud broker architectures to conduct the needed services selection, optimization and management. The decision making module of the broker architecture is responsible for the provider's selection. To achieve the best trade-off between the multiple QoS parameters specified in the SLA contract, the decision making module uses a multi-criteria method.

4.2 Multi-criteria algorithm for service provider's selection

Since the main advantage of adopting a cloud technology is economic, the majority of utilized methods for making decisions in a brokering context, limits the decision making to the relative cost of leased cloud resources. They do not take into account crucial key factors influencing the Quality of the Service and the client satisfaction. To address this shortcoming, we employ a multi criteria decision making method which is a qualitative comparison approach that evaluates several alternatives to find the best solution based on different conflicting criteria.

Several methods have been proposed to solve MCDM problems, and applied to different applications. Between the variety of solutions, which methods should inspire and integrate our brokering architecture? The widely used MCDM methods described in literature are:

- **The Outranking approaches:** examine if an alternative outperforms another alternative and they retain the alternative with maximum advantage and least conflict relative to diverse criteria. The most used outranking approaches are ELECTRE (90) (91) and PROMETHEE (92). For example, ELECTRE method enables to select the best choice with maximum advantage and minimum conflict in the function of the various criteria. It basically performs a pairwise comparison between the alternatives and builds an outranking relationship between them. This relationship is then used to identify and eliminate the alternatives that are dominated by other alternatives to yield a smaller set of alternatives (called the kernel).

- **Pairwise comparisons:** AHP (Analytical Hierarchy Process) (93) is one the most popular and widely used pairwise comparison methods especially in the cloud domain (94). It includes pair wise comparison of different alternatives for different criteria. It decomposes a decision problem into its constituent parts in the form of hierarchy or a set of graduated levels. Generally the hierarchy has tree levels which are the goal the criteria and the alternatives (95).

- **Multi-criteria value functions or multi-attribute utility theory (MAUT):** MAUT (96) (97) is an utility theory solving the problems of multi-objective decision making. It provides a logical mean to make trade-offs between conflicting

objectives. It assigns an utility to each criteria influencing the final decision and calculates the best global utility.

Generally, we resort to outranking models when the aggregation of criteria metrics is not easy and measurement units are different and incomparable. However, these methods are not appropriate for our problem since, in some cases, they do not propose an optimum solution (98). AHP is a flexible and intuitive methods (51), however, it could present some irregularities in ranking (51) (99). Indeed, defending the MAUT method, Luce and Raiffa (100) introduced a particular situation in which MAUT provides accurate results whereas AHP gives reversal ranking. Robert (99) explains further this situation by the following example:

"*For example, when buying a car, you first consider and rank three different ones (A,B,C) and find that A has the highest rank. You add a fourth, say an exact copy of C, and for the new problem, a ranking of the four cars now causes B to have the highest rank. MAUT proponents use this type of problem in their attack on the AHP because rank reversal can occur under AHP but not under MAUT.*"

In this work, we propose a decision making module to find the best suited service to a user demand based on the MAUT method. The main advantage of relying on this method is that the optimization problem is formalized as a single objective function, so, the best compromise solution is quickly derived as the the solution having the higher utility. Besides, this method is chosen thanks to its ability to take uncertainty into account. Indeed, our brokering solution is predicting the best service provider thanks to its monitoring module.

It predicts the utility associated to each provider's criteria based on the history of the already deployed services in the provider and monitored by the monitoring module. Since the infrastructure environment is dynamically changing, these predicted utilities remain uncertain.

4.2.1 Service selection algorithm

Multi-Attribute Utility Theory (MAUT)is a structured methodology designed to handle the trade-offs among multiple requirements. It is widely used for achieving rational decisions that reflect as closely as possible the preferences of a decision maker. The objective function of the MAUT method is formed by a set of attributes. Preferences

are determined by the utility for each attribute. Three functional forms are available: additive, multiplicative and multilinear .
As a practical guide, Keeney and Raiffa suggest that for four or more attributes the reasonable models to consider are the additive and the multiplicative (1). Since our problem concerns more than four attributes, we restrict our attention to these two forms. The additive MAUT model is appropriate only if the interaction in preferences among attributes is not considered important (101), which is not the case in the cloud context. Therefore, MAUT multiplicative form is used as a basis to built our decision making algorithm of the cloud broker.

Based on the multi-criteria method, we built an algorithm dealing with the evaluation of the different provider's offers against SLA criteria as detailed below. The cloud consumer requirements are defined by vector R:

$$\begin{aligned} R &= [r_1, r_2, \cdots, r_n] \\ n &: \ number\ of\ parameter\ in\ the\ request \end{aligned} \tag{4.1}$$

The set of providers able to respond to the user demand are represented by the vector P:

$$\begin{aligned} P &= [p_1, p_2, \cdots, p_m] \\ m &: \ number\ of\ providers \end{aligned} \tag{4.2}$$

The criteria under negotiation are given by C:

$$\begin{aligned} C &= [c_1, c_2, \cdots, c_k] \\ k &: \ number\ of\ criteria \end{aligned} \tag{4.3}$$

The user preferences are specified as weights in a vector W assigned to the criteria vector C. The weight assigned to each criterion reflects its relative importance in the customer's request:

$$W = [w_1, w_2, \cdots, w_k] \tag{4.4}$$

Once the decision parameters are initialized, the broker architecture applies the multiplicative form of the MAUT method:

$$\begin{aligned} \forall i \in [1, m], U(p_i) &= \frac{1}{w}[\prod_{j=1}^{k}(1 + ww_j f_j(c_j)) - 1)] \\ where\ (1 + w) &= \prod_{j=1}^{k}(1 + ww_j) \end{aligned} \tag{4.5}$$

The system calculates the global utility function of each alternative p_i and determines the optimal mixture that best fits the user request.

The proposed algorithm deals with choosing among a set of alternatives which are described in terms of their attributes. To provide accurate results, it requires information about:

- the user preference among the values of a given attribute expressed as utility;
- the user preference across attributes expressed as weights.

The evaluation of the weights of attributes is a question handled by several works using different MCDM methods (102) (103). In this work, we suppose that the weights are directly introduced by consumers in numeric form and we will focus principally on the utility function evaluation. Indeed, it is easier for the consumers to express their preferences toward various criteria rather than expressing their satisfaction degree relative to the offer proposed by a provider which requires technical and economical market knowledge. Using utility functions to convert numerical attribute scales to utility unit scales allows direct comparison of diverse measures.

The construction of consumers' utility functions allows to model and represent digitally their preferences. It is used in different fields especially in economics, nevertheless, there is not a common way to define it. Utility function has many forms, such as exponential curve, logarithmic curve, linear curve, hyperbolic curve etc. However, using the same curve to all criteria may not fully capture the preferences of the user.

In this work, we use utility functions to represent the degree of client satisfaction about the provider's performance of each QoS parameter, in a dimensionless scale (values from 0 to 1). Each QoS parameter is specified by a mathematical expression that determines the shape of the client satisfaction curve.

In the following, we define utility functions representing QoS metrics to compare accordingly the cloud provider's performance.

4.2.2 Functional QoS Utility functions

The SLA contract established between providers and consumers classifies the QoS negotiated metrics into different categories as illustrated by table 3.1. In the following, we will specify the utility functions related to each metric.

4.2.2.1 Compute Utility functions

Response Time Utility function The response time measures the SLA on CPU utilization so; the efficiency of a service can be quantified in terms of response time. Its utility is defined by a decreasing function as the response time increases (104, 105):

$$U_{rt} = \frac{\exp^{-SLA_{rt}+R}}{1+\exp^{-SLA_{rt}+R}} \qquad SLA_{rt} \geq 0, R \geq 0 \tag{4.6}$$

SLA_{rt} is the average response time of the cloud broker and R is the inflection point of the utility curve. The function decreases fast after the response time exceeds this value. Figure 4.1 represents an example for $R = 5s$

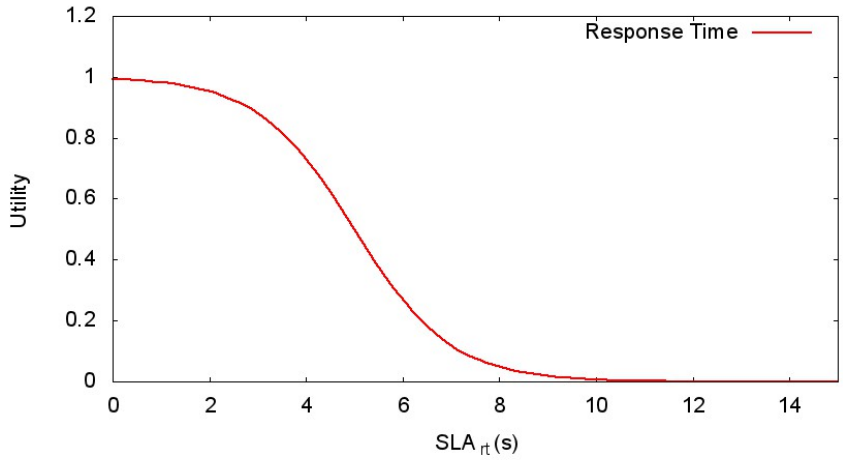

Figure 4.1: Response Time utility function -

Availability Utility function Despite the use of redundancy of crucial components and multiple deployments in different clouds, services could be inaccessible in a period

of time. Availability measures the uptime of a service in specific time interval (106).

$$\begin{aligned} Availability &= \frac{MTBF}{MTBF + MTTR} \\ MTBF &: MeanTimeBetweenFailure \\ MTTR &: MeanTimeToRepair \end{aligned} \tag{4.7}$$

The availability utility function curve is similar to Figure 4.1.

4.2.2.2 Network Utility functions

Latency Utility function The utility of a content delivery network depends on the waiting time. Latency defines an SLA of the waiting time per service transaction. The network latency could be modelled as a solution of a differential equation of the form: $A\exp^{-kt}$. This type of function is largely used in physical domain to represent the radioactive activity of a source, the discharge of a capacitor in the RC circuit, current annulations in a RL circuit ... Thus, the latency utility function is (107):

$$\begin{aligned} U_l &= c\exp^{-kSLA_l} \\ c &\in [0,1], k \geq 0 \end{aligned} \tag{4.8}$$

SLA_l is the average latency time and c is a coefficient of correlation indicating the best utility that could be achieved. The closer c is to 1; the best the utility function represents user satisfaction. Figure 4.2 shows an example of latency utility function for $c = 0.95$ and $k = 0.3$.

Bandwidth Utility function Bandwidth has several related meanings. In this work, it refers to data transfer rate. It is defined as the amount of data that can flow through a network at a given period of time. It is usually measured in bits (of data) per second (bps). Bandwidth utility function is monotonically non-decreasing (108); in other words, more bandwidth allocation should not lead to degraded application performance.

$$\begin{aligned} U_{bw} &= 1 - \exp^{-\frac{\alpha_{bw} SLA_{bw}}{bw_{max}}} \\ \alpha_{bw} &\geq 0, bw_{max} \geq 0 \end{aligned} \tag{4.9}$$

For $bw_{max} = 20Mbit/s$ and $\alpha_{bw} = 3.5$, we obtain the curve of Figure 4.3

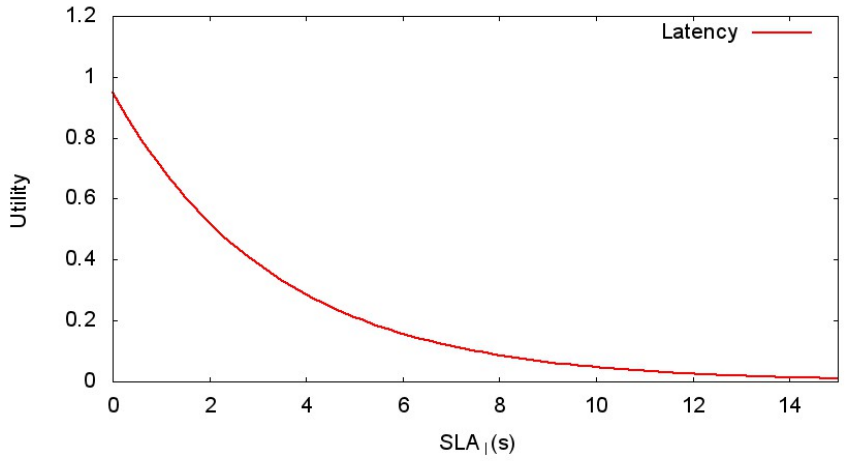

Figure 4.2: Latency utility function -

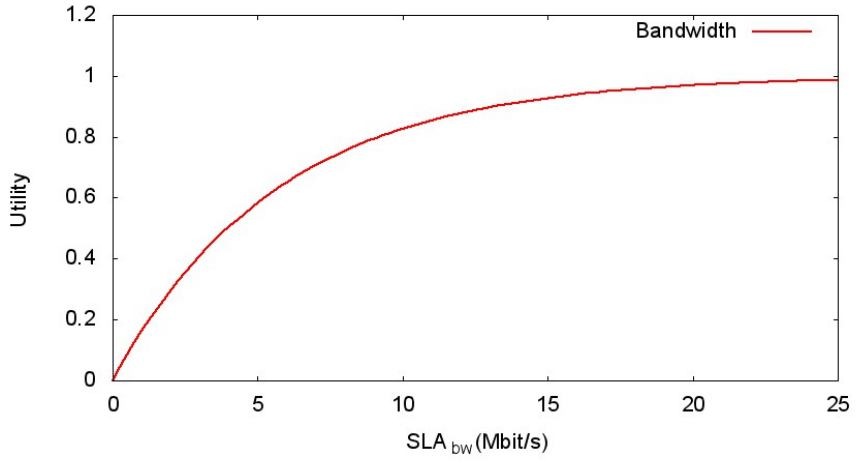

Figure 4.3: Bandwidth utility function -

4.2.2.3 Storage Utility functions

Performance Utility function Providing performance guarantees for shared storage resources is an ambiguous challenge that remains an area of active research (109). Indeed, storage performance relies on internal network performances covering latency between compute and storage. So, to evaluate the storage performance, our cloud broker focuses on the network Input/Output performance deducted from the read and write throughput. To calculate data transfer speeds, we use the equations:

$$ReadTime = \frac{DataSize}{Troughput_{read}} \tag{4.10}$$

and

$$WriteTime = \frac{DataSize}{Troughput_{write}} \tag{4.11}$$

Therefore the utility function evaluating throughput U_{bw} can be used for storage performance evaluation. The utility function U_{rt} for response time is also useful if the broker receives directly information about the read and write times.

Recovery Utility function Cloud based storage is cheaper and more scalable than installing huge databases, but recovering data after an unplanned outage, disaster or system failure can be a challenge as data is geographically distributed. Thence, to ensure uninterrupted availability of data, data fast recovery is an important SLA factor. Restore time measures the duration of time within which the database is recovered. Its utility function has the same curve as figure 4.1.

4.2.2.4 Cost Utility function

The billing systems of different cloud providers are not transparent and clear enough to compare easily prices. In deed, there is a large variations in pricing. We notice:

- Different units: for a given resource, providers may have different units of measure. For instance, CPU capacity is generally calculated in GHz however, some providers use the Xeon unit.

- Bundling/Unbundling: every provider has its own way of billing. Some ask users to fix technical characteristics like RAM, storage and data transfer rates. This billing model is used by providers like GoGrid and RackSpace. Others propose predefined bundles. This is used by a multitude of providers like Amazon EC2, Windows Azure, Google Apps and VPS.net.

- Different bundles: there is no standard for bundles. They are defined by IaaS providers differently. For example, an instance small for Amazon EC2 is equivalent to 1,7 GO of memory and 1 EC2 compute unit. However, for Windows Azure, it is 1,75 GO of memory and 1,6 GHz.

- Variations in pricing: generally, cloud providers take care that prices are convincing. Nevertheless prices of the same resources may fluctuate according to the way we allocate it. For instance, in some cases, $2 * price(smallBundle) \neq price(mediumBundle)$

In practice, there are two ways for cloud providers to answer the user request: as a set of cloud resource characteristics or as a set of bundles. Therefore, the cloud broker calculates the total price of the cloud service using a generic pricing model:

$$\begin{aligned} C_p = \sum_{i \in I_p} (x_i p_i) NBL_p + \sum_{j \in J_p} (y_j b_{p_j}) BL_p \\ p : provider \\ I_p : provider's\ resources \\ x_i : required\ number\ of\ units\ of\ each\ resource \\ p_i : unit\ price\ of\ resource \\ J_p : provider's\ bundles \\ y_j : required\ number\ of\ units\ of\ each\ bundle \\ b_{p_j} : unit\ price\ of\ the\ bundle \\ BL_p, NBL_p : dummy\ variables \\ (if\ bundling\ BL_p = 1\ else NBL_p = 1) \end{aligned} \tag{4.12}$$

For the same request, the broker calculates the service prices. To determine the utility of each one, we compare them:

- If provider's price is the lowest:

$$\begin{cases} U_p = 1 \\ P_{ref} = C_p \end{cases}$$

  where P_{ref} is the reference price on which we will build our comparison and determine the utility of the other cloud prices.

- Otherwise: $U_p = \frac{P_{ref}}{C_p}$

4.2.3 Non Functional QoS Utility functions

Non-functional properties are usually too abstract and considered as evaluation parameters after a development or deployment process. They are not addressed when requesting a cloud provider even if they are fundamental for client satisfaction. To ascertain that users needs and goals are met successfully, the cloud broker considers several non-functional properties in decision making such as reliability and trust.

4.2.3.1 Reliability Utility functions

Scalability Utility function We consider a cloud system scalable not only if it is able to easily scale under stress but also if the additional cost generated by the increased user's demand is not excessive. Accordingly, we define the proportional performance loss as follow:

$$\begin{aligned} scalability = \alpha_{sc} * \frac{RT_y - RT_x}{RT_x} \beta_{sc} * \frac{cost_y - cost_x}{cost_x} \\ \alpha_{sc}, \beta_{sc} : the\ importance\ weights\ of\ response\ time\ and\ cost\ respectively \\ RT_y : Response\ Time\ of\ new\ cluster\ Y (Y > X) \\ RT_x : Response\ Time\ of\ cluster\ X \\ cost_y : Cost\ of\ the\ cluster\ Y \\ cost_x : Cost\ of\ the\ cluster\ X \end{aligned} \tag{4.13}$$

A perfectly scalable system would have a scalability score equal to 0. Thus, utility is maximum when the proportional performance loss is null. In this case, the average response time of the new cluster did not change and there is no additional cost. However, when the proportional performance loss exceeds a certain threshold sc_{ref}, the system is no more scalable. Scalability utility function is linear as illustrated by figure 4.4.

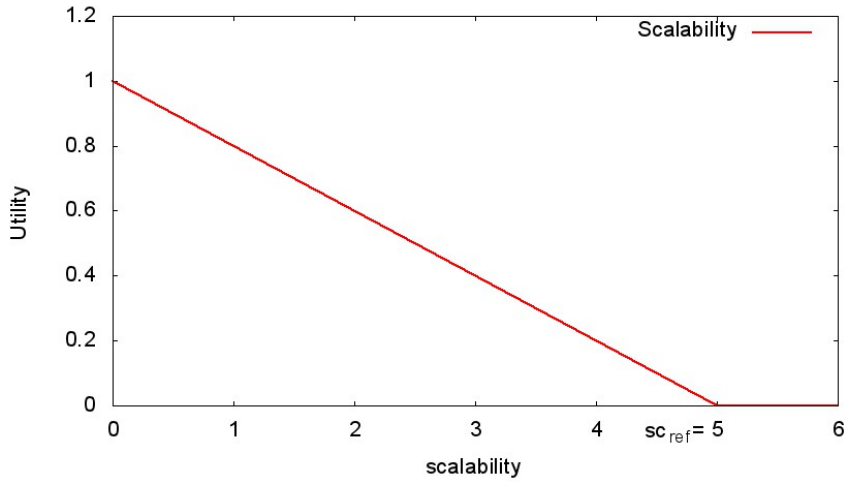

Figure 4.4: Scalability utility function -

Elasticity Utility function Elasticity is the characterization of how a cluster reacts when new nodes are added or removed under load.

$$\text{Elasticity} = \text{Upscaling} + \text{Downscaling}$$

Properties influencing elasticity are:

- Time needed for a cluster to stabilize
- Impacts on performances

Therefore, we define the elasticity's utility as follow:

$$\begin{aligned} U_{Elast} &= \frac{\alpha_{elast} * U_{rts} + \beta_{elast} * U_{Perf}}{\alpha_{elast} + \beta_{elast}} \\ U_{rts} &: utility\ related\ to\ time\ to\ stabilize \\ U_{Perf} &: utility\ related\ to\ performance \\ \alpha_{elast},\ \beta_{elast} &: importance\ weights \end{aligned} \tag{4.14}$$

U_{rts} represents the time to stabilize so it could be represented by a decreasing function with an inflection point as illustrated in figure 4.1.

U_{Perf} represents the capacity of the cloud provider to quickly provision or de-provision resources as needed. Indeed, resources allocated to an application in a period of time are seldom just equal to the demand. To illustrate the trade-off between

user's demand and provider's allocated capacity, four possible scenarios arise. The first scenario is peak-load provisioning, illustrated by figure 4.5. Cloud provider allocate resources requested by its customers without predicting any additional ones for eventual increasing needs. In this case, cloud provider can hardly scale up, but just satisfies user's requests.

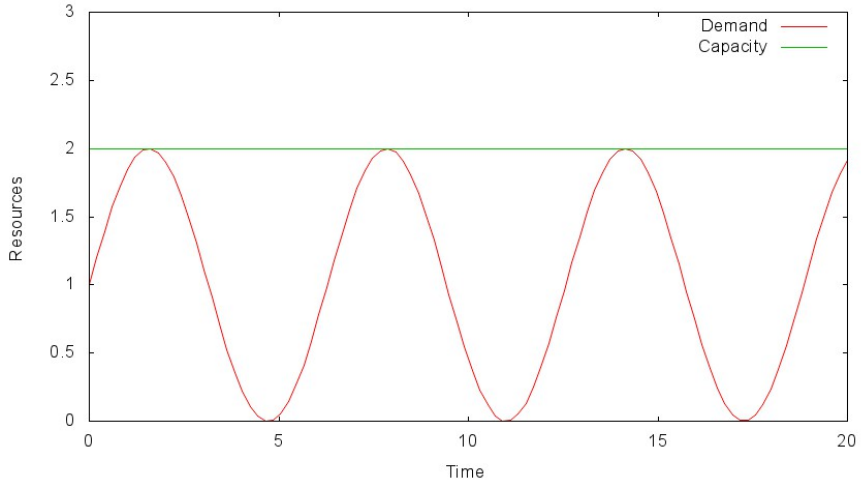

Figure 4.5: Peak-Load-provisioning scenario -

The second scenario illustrated by figure 4.6 is under-provisioning . Cloud provider fails to satisfy user's requests in peak demand. This kind of provider should be avoided because of their poor utility.

The third scenario is over-provisioning, when the provider allocates more resources than requested to overcome potential increased needs to easily scale up. In this case, the utility is good as illustrated by figure 4.7.

The fourth scenario is on-demand-provisioning (figure 4.8). Provider adapts perfectly user demand changes. It ensures perfect elasticity and it can save time and money to its customers. Its utility is high.

Weinman (110) proposed an elasticity measurement model. It represents a real demand capacity and its corresponding resource allocation both fluctuating over the time. He defines the perfect capacity strategy and calculates the loss function associated with the cost of unused or unserved resources. This model is improved by Islam and all (111). To evaluate the performances of a cloud provider, we are interested in different techniques employed by providers to best fit their allocated resources to the user demand.

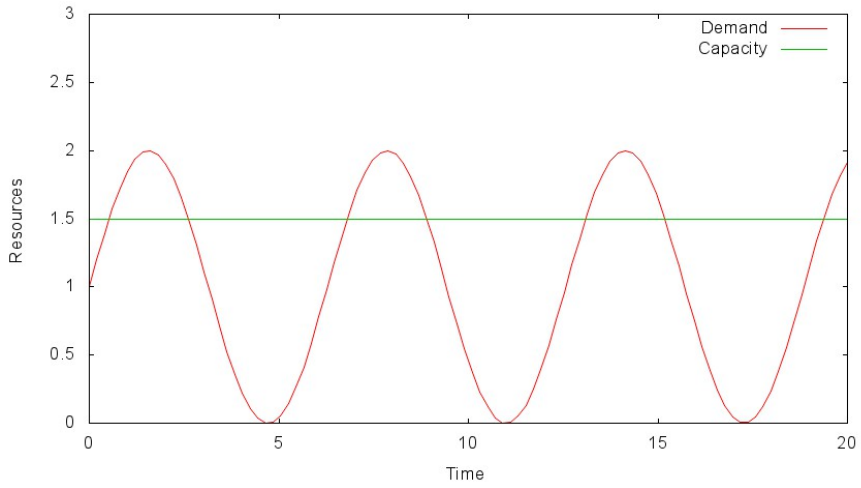

Figure 4.6: Under-provisioning scenario -

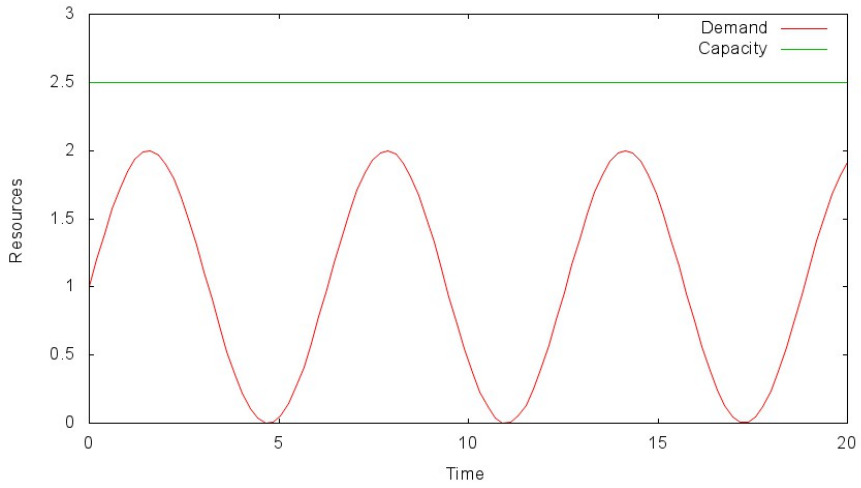

Figure 4.7: Over-provisioning scenario -

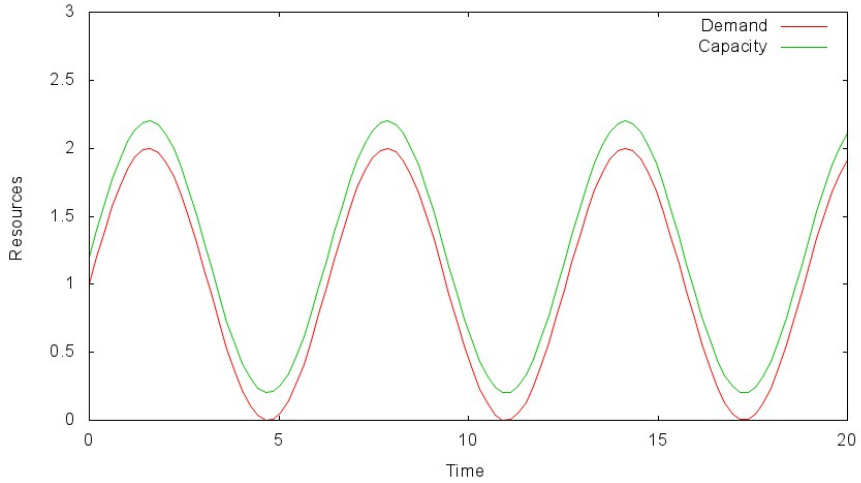

Figure 4.8: OnDemand-provisioning scenario -

Existing elasticity control strategies are either reactive or pro-active. Reactive policies are employed to regulate the amount of allocated resources after detecting an over or under provisioning. Pro-active policies, like history based predictive policies or expert knowledge based predictive policies, are likely to prevent demand fluctuation and to act consequently. We define the elasticity control performance as follow:

$$U_{Perf} = \frac{\gamma E_{react} + \delta E_{proact}}{\gamma + \delta}$$

$$E_{react} : efficiency\ of\ reactive\ elasticity\ control\ policies\ used\ by\ the\ provider$$

$$E_{proact} : efficiency\ of\ proactive\ elasticity\ control\ policies\ used\ by\ the\ provider$$

$$\gamma, \delta : importance\ weights$$

$$\gamma \leq \delta \tag{4.15}$$

Disaster recovery Utility function The shared nature of cloud computing environments makes them an ideal model for disaster recovery. Over time, two distinct approaches to disaster recovery models emerged:

- Dedicated model: a backup infrastructure is dedicated to a single user. Its hardware and software are preconfigured so it is ready to run as soon as it receives the backup data image. Thence, this model is recommended for a rapid recovery

time nevertheless it is costly because the hardware sits idle when not being used for disaster recovery.

- Shared model: a backup infrastructure is shared among multiple users. This second alternative is favorable for a reduced cost but its recovery time is slower.

Table 4.1: Disaster recovery models.

	Cost	**Speed to recovery**
Dedicated Model	↗	↗
Shared Model	↘	↘

Cloud-based business resilience provides an attractive alternative to traditional disaster recovery, offering both the shorter downtime associated with a dedicated infrastructure and the reduced capital expenses that are consistent with a shared recovery model. To evaluate the effectiveness of a disaster recovery approach in a cloud context, two recovery objectives are measured:

- Recovery time objective (RTO): How long to recover? The amount of time needed to start the restoration process.

- Recovery point objective (RPO): How much data is lost? Minimizing data loss is an important objective of a successful disaster recovery solution. So, we calculate the amount of data lost during outage.

Based on factors mentioned above, we measure the recovery performance as follow:

$$\begin{aligned} Recov_{perf} &= \alpha * RTO + \beta * RPO \\ Recov_{perf} &= \alpha * \frac{T_F}{T_S + T_F} + \beta * \frac{D_L}{D_S + D_L} \\ T_F &: duration\ of\ failure \\ T_S &: data\ loss \\ D_S &: data\ stored \end{aligned} \tag{4.16}$$

Additional cost generated by an outage is an important factor to evaluate a disaster recovery strategy. We measure recovery cost as follow:

$$\begin{aligned} Recov_{cost} &= \frac{Cost_F}{Cost_S + Cost_F} \\ Cost_F &: additional\ cost\ of\ failure \\ Cost_S &: cost\ of\ cloud\ services \end{aligned} \tag{4.17}$$

Therefore, disaster recovery utility is:

$$\begin{aligned} U_{recov} &= \frac{\alpha * U_{recovPerf} + \beta * U_{recovCost}}{\alpha + \beta} \\ U_{recovPerf} &: linear\ utility\ of\ recovery\ performances \\ U_{recovCost} &: linear\ utility\ of\ recovery\ cost \\ \alpha, \beta &: importance\ weights \end{aligned} \tag{4.18}$$

To be better adapted to user needs, the cloud broker distinguishes between dedicated model and shared model scenarios. In dedicated model, users devote more budgets to recovery but achieve rapid and efficient data recovery as illustrated by figure 4.9.

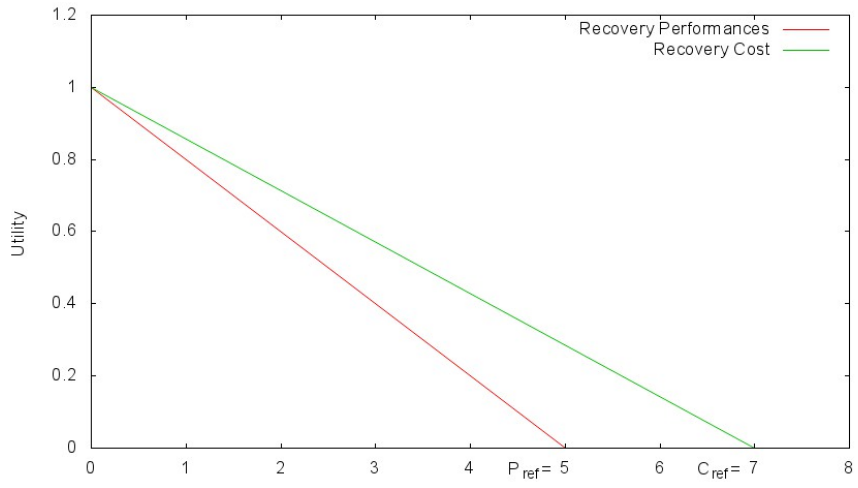

Figure 4.9: Dedicated recovery model scenario -

However, in shared scenario (figure 4.10), users need to spend minimum cost for recovery and tolerate some degradation in performance. In this case, performance recovery utility and cost recovery utility would be different from the dedicated scenario as thresholds of performance and cost measures are different.

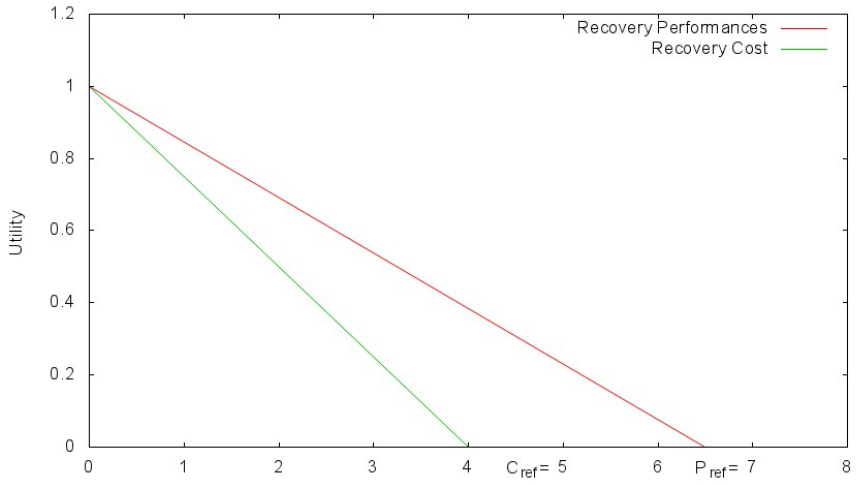

Figure 4.10: Shared recovery model scenario -

4.2.3.2 Trust Utility function

Trust and reputation systems are successfully used in numerous application scenarios to support users identifying the reliable and trustworthy providers. Our designed cloud broker is supporting customers to select the appropriate cloud provider. We argue that trust should be taken into consideration for service provider selection.

Cloud model (112) (113) is an exchange model of qualitative and quantitative, uniformly describing randomness, fuzziness and their relationship. It mainly reflects the two uncertainties in the event of universe or concepts in human knowledge: fuzziness and randomness. The general concept of cloud model can be expressed by its three numerical characteristics (Ex,En,He):

- Expectation (Ex): the point that can best represent a qualitative concept in the domain space. Ex represents the expectation of trust by quantifying the concept in a representative sample points.
- Entropy (En): it reflects the uncertainty of qualitative concept and represents its granularity (the ambiguity). It is a randomness measure of the quantitative concept.
- Hyper Entropy (He): measure of uncertainty of entropy (the entropy of entropy). It is a measure of the dispersion on the cloud Droplets.

Obviously, entities with higher trust and lower uncertainty are more trustworthy, while those with lower trust value and higher uncertainty are not trustworthy. The most important algorithms (114) in cloud model are:

- Normal Cloud Generator Algorithm:
  - Input: (Ex,En,He)
  - Output: Cloud Droplets

  Used to generate the required number of Cloud Droplets when knowing three characteristics figures (Ex,En,He). But what is a cloud droplet?

  Cloud is composed by many cloud droplets, each one is a point characterizing cloud in the domain space. We consider U: a quantitative domain with numerical representation if $x \in U$ then $u_A(x) \in [0,1]$ $u : U \to [0,1]$ $x \mapsto u_A(x)$ x is called a Cloud Droplet and the distribution of x in the domain of U is called Cloud.

- Backward Generator Algorithm
  - Input: N Cloud Droplets x_i $(1 \leq i \leq N)$
  - Output: Ex,En and He of the N cloud Droplets

  Obtains tree positive characteristics figures (Ex,En,He) of the Normal Cloud Generator from a set of given Cloud Droplets. Sample to achieve qualitative evaluation of sample data.

The cloud trust model is complex and could be hardly introduced in our utility model to evaluate providers trustworthiness. Thus, we are interested in a novel model for the evaluation of trustworthiness of complex systems called CertainLogic (115). This model has been already tested in the context of cloud computing (116). CertainTrust (CT) is designed as a representation for evidence-based trust, but may also serve as a representation for uncertain probabilities. Systems can be illustrated by graphs and their trustworthiness is evaluated based on standard operators of propositional logic introduced by the authors. CT introduces a novel way for modeling probabilities and uncertainties:

For a proposition A, CT define o_A the opinion about its truth. Each opinion is modelled as a triple of values, $o_A = (t, c, f) \in [0,1]x[0,1]x[0,1]$ where

- t denotes the average rating: the degree to which past observations support the truth of the proposition.
- c denotes the certainty associated with the average rating: the degree to which the average rating is assumed to be representative to the future. In case of complete uncertainty (c=0), the expectation value depends only on f.
- f denotes the initial expectation assigned to the truth of the statement: the assumption about the truth of a proposition in absence of evidence.

Taking into account all those factors, the truth of a proposition A is defined as:
$E(t, c, f) = t * c + (1 - c) * f$
CL defines logical operators (OR, AND and NOT) to evaluate the trustworthiness of a system from the trust values of its atomic components. Figure 4.11 shows the definition of logical operators as introduced by (115).

OR	$c_{A\vee B} = c_A + c_B - c_A c_B - \frac{c_A(1-c_B)f_B(1-t_A) + (1-c_A)c_B f_A(1-t_B)}{f_A + f_B - f_A f_B}$ $t_{A\vee B} = \begin{cases} \frac{1}{c_{A\vee B}}(c_A t_A + c_B t_B - c_A c_B t_A t_B) & \text{if } c_{A\vee B} \neq 0\ , \\ 0.5 & \text{else}\ . \end{cases}$ $f_{A\vee B} = f_A + f_B - f_A f_B$
AND	$c_{A\wedge B} = c_A + c_B - c_A c_B - \frac{(1-c_A)\,c_B\,(1-f_A)\,t_B + c_A\,(1-c_B)\,(1-f_B)\,t_A}{1 - f_A f_B}$ $t_{A\wedge B} = \begin{cases} \frac{1}{c_{A\wedge B}}\left(c_A c_B t_A t_B + \frac{c_A(1-c_B)(1-f_A)f_B t_A + (1-c_A)c_B f_A(1-f_B)t_B}{1-f_A f_B}\right) & \text{if } c_{A\wedge B} \neq 0, \\ 0.5 & \text{else}\ . \end{cases}$ $f_{A\wedge B} = f_A f_B$
NOT	$t_{\neg A} = 1 - t_A$, $c_{\neg A} = c_A$, and $f_{\neg A} = 1 - f_A$

Figure 4.11: CertainLogic operator's definition -

In this work, we will focus on the evaluation of trusted cloud providers in order to help customers select trustworthy ones. The trustworthiness of a cloud provider depends on the expected behavior of the services and underlying systems with respect to specific attributes:

- Security: It is still a matter of great concern for a cloud user to trust security of cloud services. All the security techniques are built on confidentiality, integrity and availability.

  - Confidentiality: is achieved through encryption. The primary purpose of cryptography is information management. If the data is confidential, it cannot be read or understood by anyone other than the intended recipient or recipients. Use encryption to protect sensitive data that is contained in a message. Unencrypted data, which is known as plaintext, is converted to encrypted data, which is known as ciphertext. Data is encrypted with an algorithm and a cryptographic key. Ciphertext is then converted back to plaintext at its destination.
  - Integrity: means that data cannot be modified undetectably. Only authorized users can access or modify information. So, Integrity is violated when a message is actively modified in transit. Measures are taken to ensure integrity includes controlling the physical environment of networked terminals and servers, restricting access to data, and maintaining rigorous authentication practices. Authentication is the verification of the identity of a person or process. It verifies that messages really come from their stated source, like the signature on a (paper) letter. The most common form of authentication is typing a user name (which may be widely known or easily guessable) and a corresponding password that is presumed to be known only to the individual being authenticated.

- Availability: is the degree to which system or component is operational and accessible when required for use (see paragraph 4.2.2.1). Uptime and downtime are characteristics measuring availability. For instance, a cloud provider warranting seven nines uptime availability, have almost 0.3 second of downtime per year.

- Scalability:

  - Vertical scaling (scale up): is adding more resources to the same computing pool. For example, to scale up an application running on a virtual machine, cloud providers add more processors and storage to that machine or simply move the application to a new one, more powerful. Vertical scaling is a quick and easy way to achieve scalability and it is the best solution for applications that can only run on a single machine, however, it is expensive.

  - Horizontal scaling (scale out): is the addition of more machines or devices to the computing platform. Generally, it is easier to achieve good horizontal scalability than good vertical scalability and it is largely cheaper. Horizontal scaling is dedicated to multi-tier applications and can add complexity to the system. However, it ensures more flexibility because it enables managers to grow servers and storage separately.

- Latency (see paragraph 4.2.2.2)

- Breaches: the architecture of the designed cloud broker include a rule engine module receiving customer and infrastructure feedback. We are using that information to determine the most trustworthy cloud providers.

Chapter 5

Experimentations and results

In this work, we proposed a cloud broker architecture able to find the best provider satisfying a user request based on its preferences. To ensure a better understanding between the different actors of the cloud, the broker uses semantic annotations. It uses ontology and Rule Language to define policies governing infrastructure behavior (Chapter 3). Design question of intelligent autonomous cloud broker are answered by using multi criteria decision making. We used the MCDM method MAUT (Multi Attributes Utility theory method) to determine the best agreement (Chapter 4). We modelled utility functions to evaluate user's satisfaction degree regarding functional (see Section 4.2.2) and non functional (see Section 4.2.3) criteria under negotiation.

For the evaluation of our solution, we have implemented it using java framework for cloud broker encoding and jena libraries for semantic representations. We used Prot*égé* for creating the SLA contract ontology which is the basic semantic model for information exchange in the proposed solution.

To confirm the advantage of the proposed architecture, three experiments have been carried out: (1) the validation of the semantic annotations contributions (2) the evaluation of the proposed functional utility functions benefits comparing to other forms of utilities and (3) the estimation of the non functional utilities attributes to be integrated in the decision making algorithm. This chapter demonstrates and discusses these results.

5.1 Validation of semantic annotations contribution

The scenario chosen for demonstration is illustrated by figure 5.1. There, a cloud customer tries to find the best IaaS virtual resources offer satisfying his request. To answer to user request, providers use different terminologies to define their cloud resource characteristics which makes the comparison between them difficult. For example, the first provider uses a french language for its resources description. To describe its cloud servers, it uses Amazon ECU instances (EC2 Compute Units). One ECU is equivalent to 1.0-1.2 GHz. The second provider can respond to the user request but uses a taxonomy that is different from the user request. Its infrastructure resources is composed by E5520 Intel Xeon processors which means that 1 CPU-core proposed by this provider is almost equivalent to 4.52 GHz. Finally, the third provider propose Intel Pentium 4 Processors with a capacity of 1.6 GHz.

Figure 5.1: Semantic evaluation scenario -

To evaluate the impact of the introduced semantic annotations, we have conducted two experimentations to calculate the utility of each parameter involved in the SLA contract negotiation. In the first one we have performed a syntactic search to compare

the needed characteristics to the provider's offers. The obtained results are illustrated by figure 5.2. The evaluation of the different providers is principally based on the CPU parameter since it is the only common parameter between the request and all the providers. However, the evaluation of this parameter is not necessary convincing. Indeed, it assigns the same CPU utility to Provider 2 and Provider 3 because they both offer two CPU cores. However, it ignores that the proposed cores do not have the same speed. Besides, we remark that provider 3 has the best final score since we detected a second parameter satisfied namely the delay. Unfortunately, this parameter corresponds to the delivery packet loss not to the latency needed by the user, which leads the consumer to choose a provider not meeting his requirements. Indeed, applying syntactic evaluation, we find that the providers are classified as follow $Utility(P3) > Utility(P1) > Utility(P2)$. However, taking into account the semantics of the attributes, we find that $Utility(P2) > Utility(P1) > Utility(P3)$ as illustrated by figure 5.3.

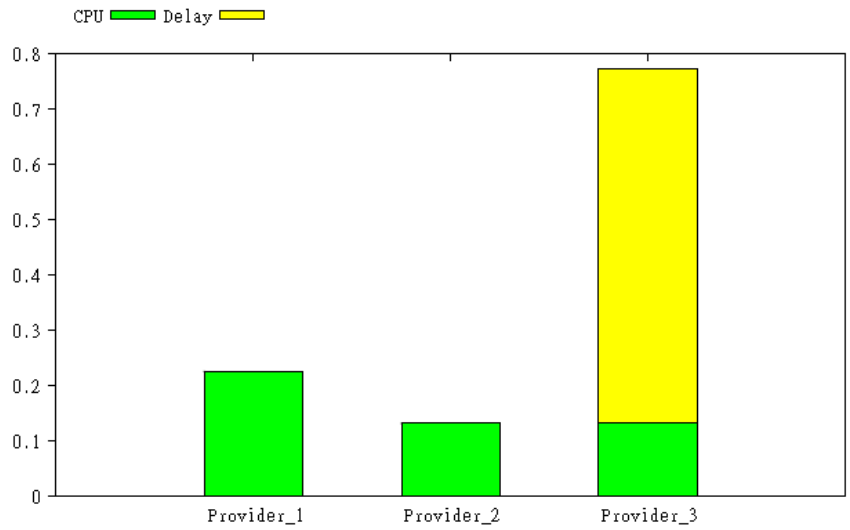

Figure 5.2: Syntactic search evaluation -

The introduction of semantic annotations enables hiding syntactic heterogeneity and allow cloud customers to discover the different provided service offers. Hence, thanks to semantic annotations, customers are not limited any-more to providers using the same taxonomy and they can avoid choosing bad proposals because of a misunderstanding of an offer. However, semantics does not help users to select the best trade off satisfying

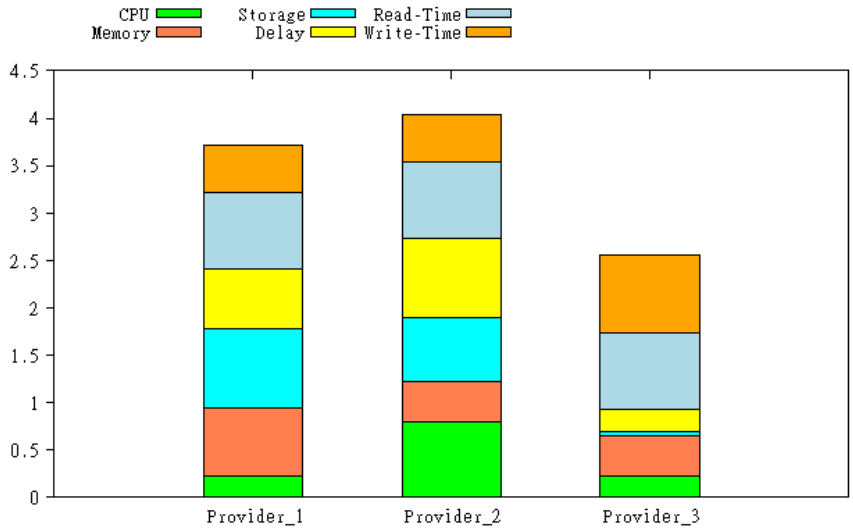

Figure 5.3: Semantic search evaluation -

their requirements and can not take their preferences into account. To meet this need, the decision making module of the cloud broker uses a multi criteria method which is the multi attribute utility theory. To better use of this method, we have customized the utility functions for the cloud. The proposed utility functions are evaluated in the next section.

5.2 Evaluation of the proposed utility functions

To evaluate provider's performance against heterogeneous criteria, we adopted the multi-criteria decision making method for selecting the best provider associated with utility function descriptions to determine the degree of user satisfaction. We modelled a large number of functional and non functional criteria.

5.2.1 Evaluation of functional utility functions

Utility function has many forms, such as exponential curve, logarithmic curve, linear curve, hyperbolic curve etc. In this work , we personalized utility functions curves to closely represent user satisfaction degree of each criteria. In the previous chapter, we have presented utility functions predefined by broker to determine user satisfaction. The described utilities have to be adjusted to the user needs whenever the broker treats a

new request. Indeed, the broker utilizes the user SLA criteria to determine the variables responsible for defining the shapes of the utility functions curves. This way, the broker determines precisely the utility of each provider SLA criteria. Table 5.1 summarizes functional SLA criteria and related parameters provided by each entity.

Table 5.1: Functional SLA criteria parameters.

	Consumer	**Broker**	**Provider**
Response Time	maxResponseTime	R	SLA_{rt}
Availability	minAvailability	R	MTBF, MTTR
Latency	maxLatency	c,k	SLA_l
Bandwidth	minBandwidth	α bw_{max}	SLA_{bw}
Read Performances	minReadTime minDiskCapacity	α bw_{max}	Th_{read}
Write Performances	minWriteTime minDiskCapacity	α bw_{max}	Th_{write}
Recovery	maxRestoreTime	R	$SLA_{restore}$

We simulated a scenario in which the cloud broker receives a user request with response time, latency and bandwidth requirements depicted by Table 5.2.

Table 5.2: User request example.

SLA Criteria	**Value**
Response Time	2 seconds
Latency	3 seconds
Bandwidth	10 Mbit/s

This request could be answered by ten different providers as illustrated by table 5.3.

As can be seen, it is difficult and time consuming to choose manually the best offer since proposed values are very close to each other. Besides, providers proposing the best values of one criteria are not necessarily as good for the other ones. For example, provider 9 suggests 1.5 seconds of response time which is the best offer for this request. However, its proposed bandwidth (9 Mbit/s) is the worse. So, how could users select the best offer?

Table 5.3: Provider's offers.

	Response Time	Latency	Bandwidth
Provider 1	1.9	2.9	11
Provider 2	2.5	3	12
Provider 3	2.2	2.8	12
Provider 4	1.6	2.6	9
Provider 5	2.2	3.2	10
Provider 6	2.0	3.1	11
Provider 7	1.9	2.8	10
Provider 8	2.1	2.9	12
Provider 9	1.5	2.7	9
Provider 10	1.8	2.6	10

To select the best offer, our cloud broker uses the multiplicative form of a multi-attribute utility theory method defined by the equation below:

$$\begin{aligned} \forall i \in [1,m], U(p_i) &= \frac{1}{w}[\prod_{j=1}^{k}(1+ww_j f_j(c_j))-1)] \\ where\ (1+w) &= \prod_{j=1}^{k}(1+ww_j) \end{aligned} \tag{5.1}$$

We suppose that the user have not particular preferences, then the weight attributed to each criteria is equal to 0.5. For w1=w2=w3=0.5, W= -0.76. In this section, we will compare the results obtained by using the MAUT method with three forms of utility functions: our proposed utilities, the linear form and the exponential form. The first step the broker uses to answer the user request is to configure its decision making module to adjust every utility function to consumer's preferences. In the following, we will explain how the utilities are configured for each negotiated criterion.

5.2.1.1 Utility functions Configuration

Response Time Utility function According to the user request, certain points are defined to configure the utility functions curves. Table 5.2 shows that the consumer needs a response time less than 2 seconds. So, we assign to this value an utility equal

to 0.8 since 2 seconds is a measure satisfying the user request but it could be better. The best measure for the response time is 0 second ie the user receives instantly the provider response and we suppose that the worst measure is 12 seconds.

In award, $\begin{cases} f(0) = 1 \\ f(2) = 0.8 \\ f(12) = 0 \end{cases}$

These values enable the setting of the linear, the exponential and our proposed utilities for the cloud broker architecture (CBA utilities).

- Linear Utilities

  Response time is a decreasing curve of the form $f(x) = \frac{x^- - x}{x^- - x^+}$ where x^- is the worse response time value and x^+ is the best value. In our case $f(x) = \frac{12-x}{12}$

- Exponential Utilities

  The exponential curve is the most widely used form for utility function. In this work, we will compare our utilities to exponential utilities represented by (102) as follows: $f(x) = a - b * exp(-cx)$ To determine the shape of the exponential curve of response time utilities, we need to resolve the following system of equations:

  $\begin{cases} f(0) = 1 \Longleftrightarrow a - b * exp(-c * 0) = 1 \\ f(2) = 0.8 \Longleftrightarrow a - b * exp(-c * 2) = 0.8 \\ f(12) = 0 \Longleftrightarrow a - b * exp(-c * 12) = 0 \end{cases}$

  We obtain: $\begin{cases} a = -1.7218 \\ b = -2.7218 \\ c = 0.038 \end{cases}$

  So, $f(x) = -1.7218 + 2.7218 * exp(-0.038 * x)$

- CBA Utilities

  In the previous chapter, we have defined the response time utility function as follows: $U_{rt} = \frac{\exp^{-SLA_{rt}+R}}{1+\exp^{-SLA_{rt}+R}}$ where SLA_{rt} is the response time criteria proposed by the provider and R is an inflexion point precising the shape of utility function. To adjust response time utility function to user requirement, we have to determine R such as $f(maxResponseTime) = U_{min}$, where $maxResponseTime$ is the minimum response time accepted by the user and U_{min} is its corresponding utility. We obtain:

$$R = maxResponseTime + log\frac{U_{min}}{1 - U_{min}} \tag{5.2}$$

Proof: Find $R/U_{rt} = U_{min}$

For $SLA_{rt} = maxResponseTime \Rightarrow U_{rt} = U_{min}$

$U_{rt} = U_{min}$

$\Rightarrow \frac{\exp^{-maxResponseTime+R}}{1+\exp^{-maxResponseTime+R}} = U_{min}$

$\Rightarrow \exp^{-maxResponseTime+R} = U_{min} + U_{min} \exp^{-maxResponseTime+R}$

$\Rightarrow (1 - U_{min}) \exp^{-maxResponseTime+R} = U_{min}$

$\Rightarrow \exp^{-maxResponseTime+R} = \frac{U_{min}}{1-U_{min}}$

$\Rightarrow -maxResponseTime + R = log(\frac{U_{min}}{1-U_{min}})$

$\Rightarrow R = maxResponseTime + log(\frac{U_{min}}{1-U_{min}})$

For our example, we obtain $R = 3.39$, hence $f(x) = exp(-x+3.39)/(1+exp(-x+3.39))$.

The different representations of response time utility are illustrated by figure 5.4.

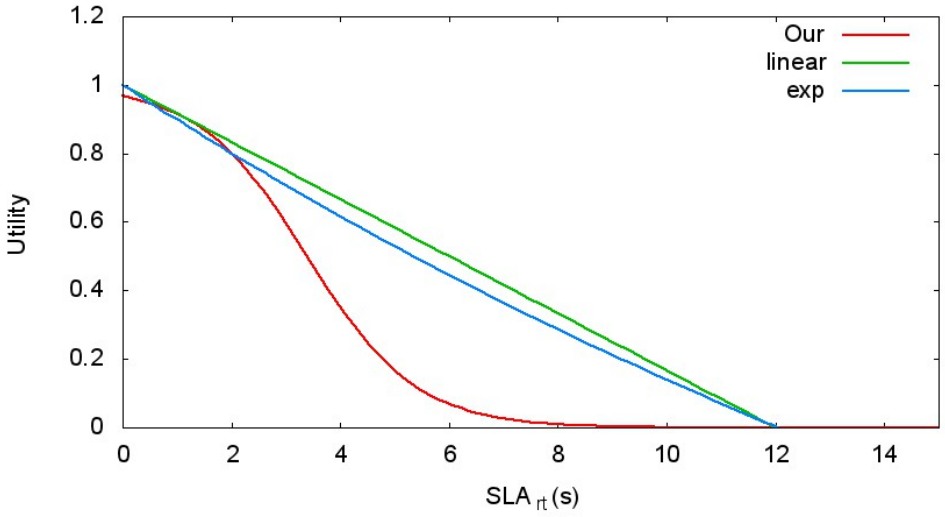

Figure 5.4: Response Time Utility functions -

In our representation of response time, we suppose that the maximum response time tolerated by the user represents a hard constraints. Indeed, we note that for CBA representation the utility drops significantly when the response time exceeds the two seconds fixed by the user request. However the linear and exponential curves represent a smooth decreasing shape which does not reflect the user constraint.

Latency Utility function For latency, to configure our utilities, we have the following points: $\begin{cases} f(0) = 1 \\ f(3) = 0.8 \\ f(60) = 0 \end{cases}$

- Linear Utilities

  Latency is also a decreasing curve of the form $f(x) = \frac{x^- - x}{x^- - x^+}$ where x^- is the worse response time value and x^+ is the best value. So, $f(x) = \frac{60-x}{60}$

- Exponential Utilities To determine the shape of the exponential curve of latency utilities, we have to resolve the following system of equations:
  $\begin{cases} f(0) = 1 \Longleftrightarrow a - b * exp(-c * 0) = 1 \\ f(3) = 0.8 \Longleftrightarrow a - b * exp(-c * 3) = 0.8 \\ f(60) = 0 \Longleftrightarrow a - b * exp(-c * 60) = 0 \end{cases}$
  We obtain: $\begin{cases} a = -0.0124 \\ b = -1.0124 \\ c = 0.0733 \end{cases}$
  So, $f(x) = -0.0124 + 1.0124 * exp(-0.0733 * x)$

- CBA Utilities

  Latency utility function is $U_l = c\exp^{-kSLA_l}$ where SLA_l is the latency SLA criterion given by cloud provider and c and k are constants to be adjusted by the broker. c is the coefficient of correlation representing the maximum utility when the latency time is null ie for $SLA_l = 0, U_l = c$. Therefore, $c = U_{l_{max}}$. k is responsible for defining the curvature. To determine k, we solve the equation $U_l = U_{min}$ for $SLA_l = MaxLatency$:

  $U_l = U_{min}$
  $\Rightarrow c\exp^{-kSLA_l} = U_{min}$
  $\Rightarrow \exp^{-kMaxLatency} = \frac{U_{min}}{c}$
  $\Rightarrow -kMaxLatency = log\frac{U_{min}}{c}$
  $\Rightarrow k = -\frac{log\frac{U_{min}}{c}}{MaxLatency}$
  $\Rightarrow k = \frac{log\frac{c}{U_{min}}}{MaxLatency}$

  For MaxLatency=3 seconds, Utility=0.8. Thus, k=0.07. We suppose that c=0.99, we obtain finally $f(x) = 0.99 * exp(-0.07 * x)$

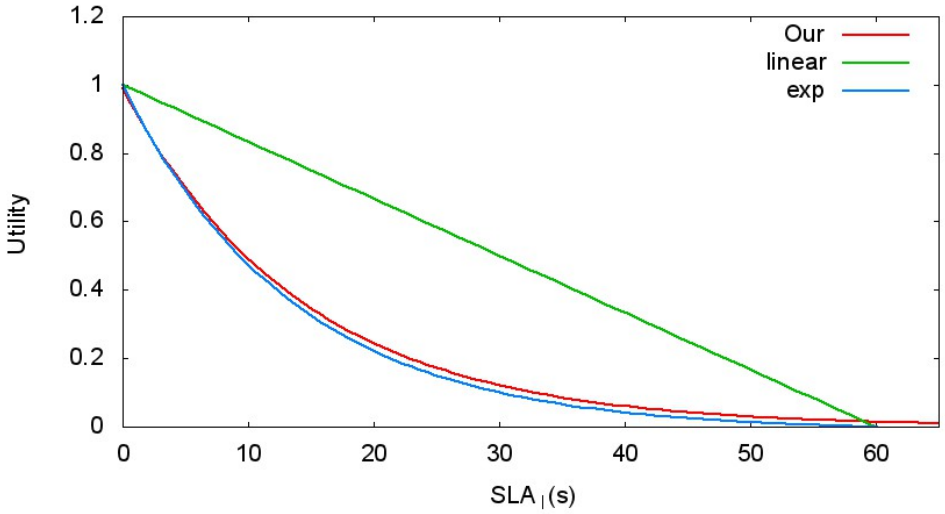

Figure 5.5: Latency Utility functions -

Figure 5.5 summarizes the tree forms of latency utility functions.

We perceive that the linear form does not represent well the latency performance. It gives higher utilities than exponential and CBA utilities which can alter the final results. However, CBA and exponential utilities are very similar especially for values close to 3 seconds requested by the user.

Bandwidth Utility function To configure bandwidth utilities, we rely on user preferences and we obtain the following system of points: $\begin{cases} f(0) = 0 \\ f(10) = 0.8 \\ f(20) = 1 \end{cases}$ We assign the utility of 0.8 to the measure of 10 Mbit/s since it is the value requested by the user. We assume that 20 Mbit/s is the maximum bandwidth that could be achieved and evidently the utility is null for 0 bandwidth received.

- Linear Utilities

  Bandwidth is a rising characteristic curve of the form $f(x) = \frac{x-x^-}{x^+-x^-}$ where x^- is the worse value and x^+ is the best. , $f(x) = \frac{x}{20}$

- Exponential Utilities

  Three equations are needed to calculate a,b and c and determine consequently the shape of the exponential curve of bandwidth utilities. Based on the values picked

previously, we have: $\begin{cases} f(0) = 0 \Longleftrightarrow a - b * exp(-c * 0) = 0 \\ f(10) = 0.8 \Longleftrightarrow a - b * exp(-c * 10) = 0.8 \\ f(20) = 1 \Longleftrightarrow a - b * exp(-c * 20) = 1 \end{cases}$

We obtain: $\begin{cases} a = 1.0666 \\ b = 1.0666 \\ c = 0.138 \end{cases}$

So, $f(x) = 1.0666 - 1.0666 * exp(-0.138 * x)$

- CBA Utilities

  The proposed bandwidth utility function equation is: $U_{bw} = 1 - \exp^{-\frac{\alpha_{bw} SLA_{bw}}{bw_{max}}}$. The broker defines the maximum bandwidth rate enabled by technology bw_{max}. The second constant that the broker has to fix is α. By formulating his request, the cloud customer provides the broker with the minimum bandwidth rate satisfying his request"minBandwidth". The broker uses this value to solve the equation $U_bw = U_{min}$ and find α as follow:

  $ForSLA_{bw} = minBandwidth, U_{bw} = U_{min}$

  $U_{bw} = U_{min}$

  $\Rightarrow 1 - \exp^{-\frac{\alpha_{bw} minBandwidth}{bw_{max}}} = U_{min}$

  $\Rightarrow \exp^{-\frac{\alpha_{bw} minBandwidth}{bw_{max}}} = 1 - U_{min}$

  $\Rightarrow \alpha_{bw} \frac{minBandwidth}{bw_{max}} = -log(1 - U_{min})$

  $\Rightarrow \alpha_{bw} = \frac{bw_{max}}{minBandwidth} log(\frac{1}{1-U_{min}})$

  In this example, $U_{min} = 0.8$ and $\alpha_{bw} = 3.2188$. Consequently, $f(x) = 1 - exp(-3.2188 * x/20)$.

Figure 5.6 summarizes the three forms of bandwidth utility functions.

We observe again that the linear form is not as good as other forms to represent bandwidth criteria. On the other hand, CBA and exponential representations provide similar utilities for values smaller than 10 MBit/s requested by the user. However, CBA utility attributes lower values to offers exceeding the user demand. This moderation is intended to avoid the compensation between attributes that influences the global utility.

5.2.1.2 Results

To find the best provider satisfying the user request defined in table 5.2, we run the decision making module of our cloud broker which executes the multiplicative form of

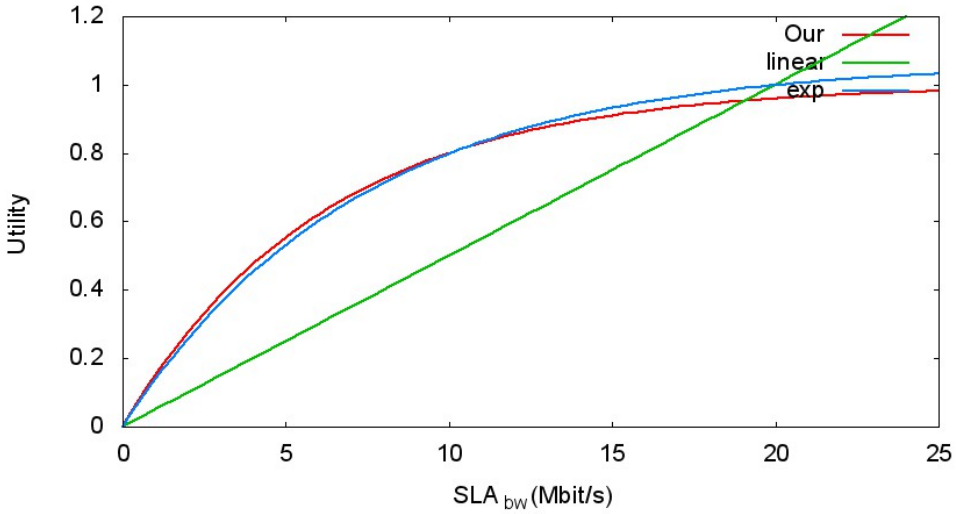

Figure 5.6: Bandwidth Utility functions -

the multi-attribute utility theory method. We compare the global utility obtained by each provider (providers are defined in table 5.3) using linear, exponential and CBA utilities. Results are illustrated by table 5.4.

We notice that the global utility attributed to each provider is different depending on the elementary utility curves used. Hence, each form of utility leads to a different classification of cloud providers as shown in table 5.5

In this experimentation, we focus on selecting the best provider when the choice is not obvious. Indeed, the provider's values chosen are very close to the user request. Hence, all providers have a good global utility as all providers obtained a global score more than 0.8 which is an acceptable value for the user. But there is no single provider that stands out due to its excellent performance. Which provider should be privileged in this case?

By using CBA utilities, the broker chooses the provider 10 while linear and exponential utilities privilege provider 8. This provider proposes the worst response time and latency values compensated by a good bandwidth offer. Moreover, we notice that the response time of provider 8 exceeds a little bit the threshold fixed by the user which could affect the user application performances. The second choice of our broker is provider 1 versus provider 3 for linear and exponential based brokers. Here, we notice that provider 3 has better bandwidth and latency performance but does not respect the threshold fixed

Table 5.4: Provider's global Utilities.

	Our	**Linear**	**Exponential**
Provider 1	0.8844	0.8638	0.8840
Provider 2	0.8639	0.8619	0.8765
Provider 3	0.8802	0.8690	0.8860
Provider 4	0.8831	0.8501	0.8777
Provider 5	0.8611	0.8451	0.8644
Provider 6	0.8780	0.8608	0.8789
Provider 7	0.8786	0.8534	0.8771
Provider 8	0.8829	0.8706	0.8868
Provider 9	0.8683	0.8519	0.8787
Provider 10	O.8849	0.8565	0.8824

Table 5.5: Provider's classification

CBA	P10	P1	P4	P8	P3	P7	P6	P9	P2	P5
Linear	P8	P3	P1	P2	P6	P10	P7	P9	P4	P5
Exponential	P8	P3	P1	P10	P6	P9	P4	P7	P2	P5

by the user for response time. In conclusion, this experimentation reveals that using specific utility curves for each criteria represents better the user request than using a generalized form such as linear and exponential forms.

5.2.2 Evaluation of non functional utility functions

Non functional properties could not be directly represented by utility functions since they are not measurable variables. Therefore, parameters such as reliability and trust are not considered when selecting a cloud provider although they are important factors that could influence the user choice (117). In this work, we present a method to quantify those parameters in order to calculate their utility for the user and integrate them subsequently in the MAUT formulas to be considered when selecting a cloud provider. In order to assessing the feasibility of our method, we will introduce some examples of non functional utilities. In the following, we will explain how to calculate the utility function of disaster recovery and trust.

5.2.2.1 Disaster recovery Example

We define disaster recovery utility by the following formulas:

$$\begin{aligned} U_{recov} &= \frac{\alpha * U_{recovPerf} + \beta * U_{recovCost}}{\alpha + \beta} \\ U_{recovPerf} &: linear\ utility\ of\ recovery\ performances \\ U_{recovCost} &: linear\ utility\ of\ recovery\ cost \\ \alpha, \beta &: importance\ weights \end{aligned} \tag{5.3}$$

To calculate the recovery performance and recovery cost utilities of a cloud provider, we have to fix thresholds of worst performance. According to the study of Alhazmi et al (118) to evaluate disaster recovery plans using the cloud, the worst recovery time noted is $RTO = 7\ days$. Besides, historically, the maximum value of data loss, calculated as the duration between two successive backups, has been $RP0 = 24\ hours$ (118). Supposing that the recovery time and the recovery data loss have the same importance, $RecovPerf_{ref} = 0.5 * 7 * 24 + 0.5 * 24 = 96\ hours = 4\ days$. To evaluate the recovery cost utility, we suppose that the maximum recovery cost is generated by the recovery of a DataWarehouse application. Wood et al (119) estimate it to 2832$ per year. The estimations are based on a "High-Memory Extra Large Instance" from EC2 which costs 3066$ per year. So, $RecovCost_{ref} = \frac{2832}{3066+2832} = 0.48$

Once $RecovPerf_{ref}$ and $RecovCost_{ref}$ are calculated, we can draw the curve of disaster recovery utility as illustrated by figure 4.9.

Let's now calculate the recovery utility of a provider proposing an $RTO = 60\ min$ and an $RP0 = 5\ min$ for a service that costs 2000$ per year and its recovery costs 500$ per year. Using equations 4.17 and 4.18 ,this provider obtains a performance utility $RecovRerfUtility = 0.9895$ and a cost utility $RecovCostUtility = 0.5833$. We assume that we are in the case of a dedicated model recovery, we attribute then more importance to the provider's performances. For $\alpha = 0.7$ and $\beta = 0.3$, this providers obtains the good utility $U_{recov} = 0.8677$.

5.2.2.2 Trust example

To determine the trustworthiness of a cloud provider, we use the CertainLogic model (115)(116) which calculates the trustworthiness of a complex system from the trustworthiness score attributed to multiple attributes characterising it. The trustworthiness of

the system is derived from the elementary scores using operators defined in figure 4.11. In this example, we suppose that the trustworthiness of a cloud provider depends on its ability to guaranty three essential attributes: security, availability and scalability. To combine an opinion on the security, we calculate the trust score of the security based on the confidentiality and the integrity scores. Since the security techniques are essentially built on confidentiality and integrity, the security score is calculated by applying an AND operator. Availability score is deducted directly from the score assigned to the uptime. Finally the scalability score results from applying an OR operation between provider's score attributed to vertical scaling and provider's score attributed to horizontal scaling. The total score assigned to a provider can be carried out by evaluating the following propositional logic term:

$(confidentiality\ AND\ integrity)\ AND$

$(availability)\ AND$

$(horizontal\ scaling\ OR\ vertical\ scaling)$

So, the trust score obtained by a provider having the capabilities illustrated by table 5.6 is $U_{trust} = 0.1861$.

Table 5.6: Factors involved in the trust score measure.

	t	**c**	**f**
Confidentiality	0.4	0.9	0.5
Integrity	0.6	0.9	0.5
Uptime	0.8	0.9	0.99
Horizontal Scaling	0.7	0.9	0.8
Vertical Scaling	0.8	0.9	0.9

The steps required to achieve this result are shown in figure 5.7.

Finally, by calculating the utility functions of non functional attributes, they may be integrated in the decision making algorithm illustrated by equation 5.1. This way, criteria such as reliability and trust could be considered in the final choice of the cloud provider.

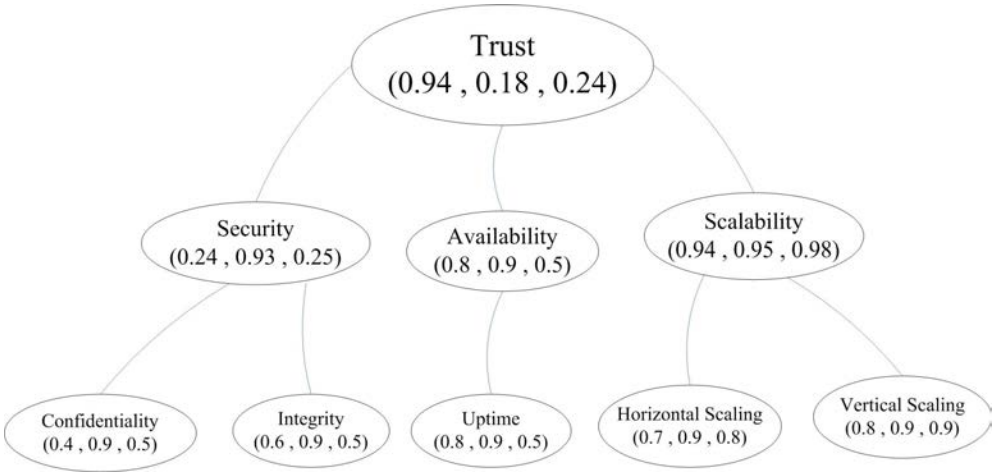

Figure 5.7: Steps to calculate the trust score -

5.3 Conclusion

This chapter evaluates the usefulness of our brokering system. It explains the efficiency of semantic annotations in hiding syntactic and semantic heterogeneity between the different actors of the cloud mainly the IaaS cloud actors. Unlike syntactic SLA languages such as WS-Agreement, our ontology enables cloud users to discover more services since it assures interoperability between providers and consumers (Section 5.1). This chapter provides also a quantitative evaluation of utility functions. It compares the functional utilities used by the broker architecture with other forms of utility representations such as linear and exponential forms. We show that the utilization of our defined utilities selects the closest provider to the user request (Section 5.2.1). We present also some examples of non functional utility measurement to explain further how non functional utilities could be estimated and integrated in the decision making algorithm (Section 5.2.2).

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Today, the cloud service provider market is very diverse. Customers can find all kind of providers, from small private clouds to large enterprises. Each of these cloud service providers might have their own set of services, business model and client base. It is then difficult for customers to know which one is a good fit for their needs. In particularly, in this work, we are interested in tree main issues encountered by cloud consumers:

- the syntactic and semantic heterogeneity of cloud services: depending on their background, the understanding of Service Level Agreements (SLAs) and what they could mean for providers and consumers is completely different. Hence, cloud actors could miss fruitful collaboration opportunities;
- the large number and the diversity of criteria involved in the choice of a cloud service provider: finding the best trade-off is not evident for customers;
- the lack of performance guarantees: customers need to supervise their application performances hosted in the provider's virtual machines and claim penalties in case of SLA contract violation.

To overcome these problems, we proposed in Chapter 3 an SLA-based brokering approach assisting cloud consumers in managing the provisioning of cloud services and selecting the best provider's offer. Our solution is based on semantic annotations in order to hide the heterogeneity between the different cloud actors. Indeed, by the use of ontologies, we defines a common vocabulary for cloud actors who need to share service

level agreement understanding in the cloud. OWL ontologies adopted help to add meaning and semantics to the data. For example, we can add a "same as" relationship between concepts having the same meaning but expressed differently by providers and consumers. We have shown that this allows to hide heterogeneity and enables cloud customers to discover more services likely to meet their requirements. Moreover, we highlighted that the use of ontology allows to automate the detection of violations in the SLA contract. This is realised by building rules that compare the real time values of the quality of service monitored with the service level objectives agreement. Thus, the reasoner executing these rules can detect violations in the SLA contract and determine the penalties to be imposed to the provider. Our cloud broker has the advantage that it can be adopted by all cloud actors using any SLA language. This is assured by the mapping between our ontology and the target SLA language. In this thesis, we have presented the example of WS-agreement language (see Section 3.5) but this task could be reproduced for any other language. This ensures interoperability between cloud actors; providers as well as consumers.

To select the best provider, which is our second problematic, we proposed the use of the decision making module in our cloud broker. We have shown in Chapter 4 how using a multi-attribute utility theory method allows us to satisfy better consumer's requirements as it enables him to find the best service provider from a multitude of providers proposals. Moreover, we think that the result of our cloud broker selection provider is more accurate compared with related works as our proposed utility functions represents better consumer preferences (better than linear equations for example as demonstrated in Section 5.2.1). Besides, in our work, we take into account both functional and non functional criteria. For these reasons, we judge that our cloud broker satisfies the best consumer requirements from functional and non functional point of view, which was never be done in the literature.

6.2 Future Research Directions

Regarding possible future works, there are some interesting activities that can be carried on.

Experimenting the cloud broker In this work, we have proposed a brokering architecture for automating the negotiation of service level agreements and the allocations of provider's resources. We have implemented and tested it using java language. We have supposed that the real time QoS values are provided by the monitoring module of our architecture. In the future, we plan to integrate our framework into a real testbed. For this purpose, we need to go over the monitoring module and propose a suitable implementation. As an example, we can use one of the several techniques used for monitoring, such as traditional server monitoring services, vendor specific monitoring services like Hyperic (76), CloudHarmony (77), Monitis (78), Nimsoft (79), Amazon CloudWatch (80) or even third party independent cloud monitoring services like Cloudstatus (81) and cloudkick (82).

On the other hand, we plan to experience the reliability of our system in a federation context. Federation is the interconnection and the collaboration of cloud providers in order to ensure load balancing. It is a relevant context to evaluate our architecture since providers face the problem of heterogeneity mentioned above and need to find the best choice allowing them to further reduce their costs.

A cross-layer cloud broker In this thesis, we have presented a generic SLA cloud ontology that we specialized to the IaaS layer (see Section 3.4). This proposed IaaS SLA ontology is conform to IaaS OCCI specification (88) as we adopted the same procedure and we used its main concepts. As a perspective to this contribution, we aim to extend our generic ontology to the other layers of the cloud i.e., PaaS and SaaS layers (see figure 2.1). To this end, we can rely on related works that try to extend OCCI specification to the PaaS and SaaS layers (120) (121). In this manner, our cloud broker can look for any service provider, whatever the involved layer. This will help users to have a sole and same entity to look for any kind of service they need.

Weight determination We have seen that a consumer may have several criteria (Quality of Service parameters) that should be satisfied in the choice of the service provider. The importance of these criteria may vary from one consumer to another and the choice of the best service provider depend on this importance (expressed by the weights in the multicriteria method we used). In other words, the result given by the broker could not be same if we change the importance of the criteria (i.e., the values

of the weights). So we should have weights that are precise enough to represent the importance of the criteria as felt by the consumer. In our work, we assumed that the weights in the multi utility attribute theory are provided by the user. In reality, it is a hard task to give these weights, especially when we have several criteria. To overcome this problematic and in order to have accurate weights, we can explore multicriteria methods that allow to determine user quality of service weights. More precisely, there is multecriteria methods that take user preferences as input to determine the weights. As an example of such methods, we can cite Analytic Hiearchy Process (AHP) (122) and Measuring Attractiveness by a Categorical Based Evaluation Technique (MACBETH) (123, 124). In this way, we think that the result of our cloud broker will be more precise and correspond the best to the consumer request.

References

[1] G.W. Torrance, M.H. Boyle, and S.P. Horwood. *Application of multi-attribute utility theory to measure social preferences for health states.* Research and working paper series. McMaster University, Faculty of Business, 1982. xv, 68

[2] IBM. **Tips for Choosing a Cloud Service Provider**. Technical report, 2011. 3

[3] Salman A. Baset. **Cloud SLAs: Present and Future**. *SIGOPS Oper. Syst. Rev.*, **46**(2):57–66, July 2012. 3, 38

[4] F. Liu, J. Tong, J. Mao, R. Bohn, J. Messina, L. Badger, and D. Leaf. *NIST Cloud Computing Reference Architecture.* Special Publication 500-292, Reading, Massachusetts, 2011. 5

[5] **Freshdesk**. `http://www.cloudreviews.com/freshdesk.html`. 8

[6] **NetSuite**. `http://www.cloudreviews.com/netsuite.html`. 8

[7] **Zoho**. `http://www.cloudreviews.com/zoho.html`. 8

[8] **Bloomfire**. `http://www.cloudreviews.com/bloomfire.html`. 8

[9] **GHG Corporation**. `http://www.cloudreviews.com/ghgcorp.html`. 8

[10] **Google App Engine**. `https://cloud.google.com/products/app-engine`. 8

[11] **AWS Elastic Beanstalk**. `http://aws.amazon.com/fr/elasticbeanstalk/`. 8

[12] **cloudmapreduce**. `https://code.google.com/p/cloudmapreduce/`. 8

[13] **Force.com**. `http://www.salesforce.com/fr/force/overview/`. 8

[14] S. Bhardwaj, L. Jain, and S. Jain. **CLOUD COMPUTING: A STUDY OF INFRASTRUCTURE AS A SERVICE (IAAS)**. *International Journal of Engineering and Information Technology*, 2010. 8

[15] **Amazon E2C**. `http://aws.amazon.com/fr/ec2/`. 8

[16] **GoGrid**. `http://www.gogrid.com/`. 8

[17] **Rackspace**. `http://www.rackspace.com/`. 8

[18] **Windows Azure**. `http://www.windowsazure.com/en-us/solutions/infrastructure/`. 8

[19] T. Forell, D. Milojicic, and V. Talwar. **Cloud Management: Challenges and Opportunities**. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 881–889, 2011. 10

[20] Kuyoro S. O., Ibikunle F., and Awodele O. **Cloud Computing Security Issues and Challenges**. *International Journal of Computer Networks (IJCN), Volume (3) : Issue (5)*, 2011. 10

[21] R. Bernnat, N. Bieber, W. Zink, and J. Strach. *Standardizing the Cloud: A Call to Action*. booz&co, 2012. 10

[22] Daniel J. Abadi. **Data management in the cloud: Limitation and Opportunities**. Technical report, 2009. 11

[23] D. Agrawal, A. El Abbadi, S. Antony, and S. Das. **Data management challenges in cloud computing infrastructures**. In *Proceedings of the 6th international conference on Databases in Networked Information Systems*, DNIS'10, pages 1–10, Berlin, Heidelberg, 2010. Springer-Verlag. 11

[24] M. Andreolini, S. Casolari, M. Colajanni, and M. Messori. **Dynamic Load Management of Virtual Machines in Cloud Architectures**. In DimiterR. Avresky, Michel Diaz, Arndt Bode, Bruno Ciciani, and Eliezer Dekel, editors, *Cloud Computing*, **34** of *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering*, pages 201–214. Springer Berlin Heidelberg, 2010. 11

[25] Y.O. Yazir, C. Matthews, R. Farahbod, S. Neville, A. Guitouni, S. Ganti, and Y. Coady. **Dynamic Resource Allocation in Computing Clouds Using Distributed Multiple Criteria Decision Analysis**. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 91–98, 2010. 11

[26] B. Sotomayor, R.S. Montero, I.M. Llorente, and I. Foster. **An Open Source Solution for Virtual Infrastructure Management in Private and Hybrid Clouds**. In *IEEE INTERNET COMPUTING, SPECIAL ISSUE ON CLOUD COMPUTING*, 2009. 12

[27] K.A. Nuaimi, N. Mohamed, M.A. Nuaimi, and J. Al-Jaroodi. **A Survey of Load Balancing in Cloud Computing: Challenges and Algorithms**. In *Network Cloud Computing and Applications (NCCA), 2012 Second Symposium on*, pages 137–142, 2012. 12

[28] V. Fusenig and A. Sharma. **Security architecture for cloud networking**. In *Computing, Networking and Communications (ICNC), 2012 International Conference on*, pages 45–49, 2012. 12

[29] E.M. Mohamed, H.S. Abdelkader, and S. El-Etriby. **Enhanced data security model for cloud computing**. In *Informatics and Systems (INFOS), 2012 8th International Conference on*, pages CC–12–CC–17, 2012. 12

[30] Gartner. **Cloud IaaS: Service-Level Agreements**. Technical report, 2011. 13

[31] M. Mohemmed Sha, I. Sherif Baig, C. Rajalakshmi, P. Balaji, and DR. K. Vivekanandhan. **WSLA Based Dynamic Monitoring and Pricing of Web Services**. *International Journal of Scientific & Engineering Research - IJSER*, 2013. 13, 38

[32] A. Keller and H. Ludwig. **The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services**. *J. Netw. Syst. Manage.*, **11**(1):57–81, March 2003. v, 14

[33] D. Davide Lamanna, J. Skene, and W. Emmerich. **SLAng: a language for defining service level agreements**. In *Distributed Computing Systems, 2003. FTDCS 2003. Proceedings. The Ninth IEEE Workshop on Future Trends of*, pages 100–106, 2003. 14

[34] G. Frankova, D. Malfatti, and M. Aiello. **Semantics and Extensions of WS-Agreement**, 2006. 15, 38

[35] I. Horrocks, B. Parsia, P. Patel-Schneider, and J. Hendler. **Semantic Web Architecture: Stack or Two Towers?** In François Fages and Sylvain Soliman, editors, *Principles and Practice of Semantic Web Reasoning*, **3703** of *Lecture Notes in Computer Science*, pages 37–41. Springer Berlin Heidelberg, 2005. v, 16, 17

[36] A. Gerber, A. van der Merwe, and A. Barnard. **A functional semantic web architecture**. In *Proceedings of the 5th European semantic web conference on The semantic web: research and applications*, ESWC'08, pages 273–287, Berlin, Heidelberg, 2008. Springer-Verlag. 16

[37] J. Bermejo. **A Simplified Guide to Create an Ontology**. Technical report, UNIVERSIDAD POLITECNICA DE MADRID, ASLab, 2007. 17

[38] R. Rosati. **On the decidability and complexity of integrating ontologies and rules**. *Web Semantics: Science, Services and Agents on the World Wide Web*, **3**(1), 2005. 17

[39] D. Fahland, D. Lübke, J. Mendling, H. Reijers, B. Weber, M. Weidlich, and S. Zugal. **Declarative versus Imperative Process Modeling Languages: The Issue of Understandability**. In Terry Halpin, John Krogstie, Selmin Nurcan, Erik Proper, Rainer Schmidt, Pnina Soffer, and Roland Ukor, editors, *Enterprise, Business-Process and Information Systems Modeling*, **29** of *Lecture Notes in Business Information Processing*, pages 353–366. Springer Berlin Heidelberg, 2009. 18

[40] S. Stoutenburg, L. Obrst, D. Nichols, P. Franklin, K. Samuel, and M. Prausa. **Ontologies and Rules for Rapid Enterprise Integration**

and Event Aggregation. In *EDOC Conference Workshop, 2007. EDOC '07. Eleventh International IEEE*, pages 173–108, 2007. v, 19, 20, 21

[41] D. Xiao and H. Xu. **An Integration of Ontology-based and Policy-based Network Management for Automation**. In *Computational Intelligence for Modelling, Control and Automation, 2006 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on*, pages 27–27, 2006. v, 21, 22

[42] D. Androcec, N. Vrcek, and J. Seva. **Cloud Computing Ontologies: A Systematic Review**. In *MOPAS 2012, The Third International Conference on Models and Ontology-based Design of Protocols, Architectures and Services*, 2012. 22

[43] T. Han and K. Mong Sim. **An Ontology-enhanced Cloud Service Discovery System**. In *International MultiConference of Engineers and Computer Scientists (IMEC 2010)*, pages 644–649, 2010. 23

[44] M. Zhang, R. Ranjan, A. Haller, D. Georgakopoulos, M. Menzel, and S. Nepal. **An ontology-based system for Cloud infrastructure services' discovery.** In *CollaborateCom*, pages 524–530. IEEE, 2012. 23

[45] Y. Ma, S. Jang, and J. Lee. **Ontology-Based Resource Management for Cloud Computing**. In NgocThanh Nguyen, Chong-Gun Kim, and Adam Janiak, editors, *Intelligent Information and Database Systems*, **6592** of *Lecture Notes in Computer Science*, pages 343–352. Springer Berlin Heidelberg, 2011. 23

[46] R. Harris. **Introduction to Decision Making**. `http://www.virtualsalt.com/crebook5.htm`, 1998. 24

[47] J. Waters, M.G. Ceruti, R. Patel, and J. Eitelberg. **Decision-Acquisition System Based on a Common Decision-Exchange Protocol**. In *Space and Naval Warfare Systems Center Pacific (SSC Pacific),15th ICCRTS, Santa Monica, CA,22-24 June, 2010*, 2010. 24

[48] S.B. Kodeswaran, O. Ratsimor, A. Joshi, and F. Perich. **Utilizing Semantic Tags for Policy Based Networking**. In *Global Telecommunications Conference, 2007. GLOBECOM '07. IEEE*, pages 1954–1958, 2007. 25

[49] L. Wu, S.K. Garg, R. Buyya, C. Chen, and S. Versteeg. **Automated SLA Negotiation Framework for Cloud Computing**. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 235–244, May 2013. 25

[50] M. Velasquez and P.T. Hester. **An Analysis of Multi-Criteria Decision Making Methodss**. *International Journal of Operations Research*, **10**(2):56–66, 2013. 26

[51] journal=American Journal of Information Systems volume=1 number=1 pages=31–43 year=2013 url=http://pubs.sciepub.com/ajis/1/1/5 doi=10.12691/ajis-1-1-5 publisher=Science and Education Publishing Aruldoss, M. and Lakshmi, T. M. and Venkatesan, V. P., title=A Survey on Multi Criteria Decision Making Methods and Its Applications. 26, 67

[52] E. K. Zavadskas, Z. Turskis, and S. Kildiene. **State of art surveys of overviews on MCDM/MADM methods**. *Technological and Economic Development of Economy*, **20**(1):165–179, 2014. 26

[53] M. Menzel, M. Schönherr, J. Nimis, and S. Tai. **(MC2)2: A Generic Decision-Making Framework and its Application to Cloud Computing**. *CoRR*, **abs/1112.1851**, 2011. 27

[54] Z. ur Rehman, F.K. Hussain, and O.K. Hussain. **Towards Multi-criteria Cloud Service Selection**. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2011 Fifth International Conference on*, pages 44–48, 2011. 27

[55] F. Liu, J. Tong, J. Mao, R. Bohn, J. Messina, L. Badger, and D. Leaf. **NIST Cloud Computing Reference Architectures**. Technical report, NIST: National Institute of Standards and Technology, 2011. 27

[56] F. Jrad, J. Tao, and A Streit. **SLA based service brokering in inter-cloud environments**. In *CLOSER(2012)76-818*, 2012. v, 28, 29, 32

[57] P. Pawluk, B. Simmons, M. Smit, M. Litoiu, and S. Mankovski. **Introducing STRATOS: A Cloud Broker Service**. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 891–898, 2012. 30

[58] **SLASOI**. http://sla-at-soi.eu/, 2008-2011. 30

[59] E. Badidi. **A Framework for Software-as-a-Service Selection and Provisioning**. *CoRR*, **abs/1306.1888**, 2013. 31

[60] **mOSAIC**. http://www.optimis-project.eu/project, 2010-2013. 31

[61] S.K. Nair, S. Porwal, T. Dimitrakos, AJ. Ferrer, J. Tordsson, T. Sharif, C. Sheridan, M. Rajarajan, and AU. Khan. **Towards Secure Cloud Bursting, Brokerage and Aggregation**. In *Web Services (ECOWS), 2010 IEEE 8th European Conference on*, pages 189–196, Dec 2010. 31

[62] **Contrail**. http://contrail-project.eu/fr, 2010-2014. 31

[63] **mOSAIC**. http://www.mosaic-cloud.eu/, 2010-2013. 31

[64] F. Moscato, R. Aversa, B. Di Martino, T. Fortis, and V. Munteanu. **An analysis of mOSAIC ontology for Cloud resources annotation**. In *Computer Science and Information Systems (FedCSIS), 2011 Federated Conference on*, pages 973–980, Sept 2011. 31

[65] **CompatibleOne**. http://www.compatibleone.org/. 32

[66] S. Yangui, IJ. Marshall, JP. Laisne, and S. Tata. **CompatibleOne: The Open Source Cloud Broker**. *Journal of Grid Computing*, **12**(1):93–109, 2014. 32

[67] A. Cuomo, G. Di Modica, S. Distefano, A. Puliafito, M. Rak, O. Tomarchio, S. Venticinque, and U. Villano. **An SLA-based Broker for Cloud Infrastructures**. *Journal of Grid Computing*, **11**(1):1–25, 2013. 32

[68] **Amazon web services**. http://aws.amazon.com/fr/. 36

[69] **Windows Azure**. http://www.windowsazure.com/fr-fr/. 36

[70] H.J. Lee, M.S. Kim, J. W. Hong, and G.H Lee. **Mapping between QoS Parameters and Network Performance Metrics for SLA monitoring**. In *Asia-Pacific Network Operations and Management Symposium*. 37

[71] M. Risch and J. Altmann. **Enabling Open Cloud Markets Through WS-Agreement Extensions**. TEMEP Discussion Papers 200920, Seoul National University; Technology Management, Economics, and Policy Program (TEMEP), 2009. 38, 56

[72] K. Kritikos and D. Plexousakis. **A Semantic QoS-Based Web Service Discovery Engine for Over-Constrained QoS Demands**. In Elisabetta Nitto and Matei Ripeanu, editors, *Service-Oriented Computing - ICSOC 2007 Workshops*, **4907** of *Lecture Notes in Computer Science*, pages 151–164. Springer Berlin Heidelberg, 2009. 38

[73] L. Wu, S.K. Garg, R. Buyya, C. Chen, and S. Versteeg. **Automated SLA Negotiation Framework for Cloud Computing**. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 235–244, May 2013. 38

[74] CSCC Workgroup. **Practical guide to cloud service level agreements**. Technical report, Cloud Standards Customer Council (CSCC), 2012. 38

[75] N. Grozev and R. Buyya. **Inter-Cloud architectures and application brokering: taxonomy and survey**. *Software: Practice and Experience*, **44**(3):369–390, 2014. 38, 39, 40

[76] **Hyperic**. `http://www.hyperic.com/`. 42, 105

[77] **CloudHarmony**. `http://cloudharmony.com/`. 42, 105

[78] **Monitis**. `http://www.monitis.com/`. 42, 105

[79] **Nimsoft**. `https://cloudmonitor.nimsoft.com/fr/`. 42, 105

[80] **Amazon Cloudwatch**. `http://aws.amazon.com/fr/cloudwatch/`. 42, 105

[81] **Entreprise Monitoring and Management of Cloud Services**. `http://www.hyperic.com/products/Cloud-monitoring.html`. 42, 105

[82] **Cloudkick Cloud Management toolkit**. `http://www.rackspace.com/cloud/monitoring/`. 42, 105

[83] **Apache Jena**. `https://jena.apache.org/`. 47

[84] G. Antoniou and F. Harmelen. **Web Ontology Language: OWL**. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 91–110. Springer Berlin Heidelberg, 2009. 47

[85] OWL Working Group. **Web Ontology Language (OWL)**. `http://www.w3.org/2001/sw/wiki/OWL`, 2012. 47

[86] B. McBride. **The Resource Description Framework (RDF) and its Vocabulary Description Language RDFS**. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 51–65. Springer Berlin Heidelberg, 2004. 47

[87] D. Allemang and J. Hendler. *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*. Elsevier Science, 2011. 47

[88] **OCCI: Open Cloud Computing Interface**. `http://occi-wg.org/`. 52, 105

[89] T. Rodrigues, P. Rosa, and J. Cardoso. **Mapping XML to Existing OWL Ontologies**. In *International Conference WWW/Internet 2006, pp. 72-77*, 2006. 56

[90] D. Bouyssou. **Outranking methods Outranking Methods**. In Christodoulos A. Floudas and Panos M. Pardalos, editors, *Encyclopedia of Optimization*, pages 2887–2893. Springer US, 2009. 66

[91] B. Roy. **The outranking approach and the foundations of electre methods**. *Theory and Decision*, **31**(1):49–73, 1991. 66

[92] J.P. Brans and B. Mareschal. **The Promethee Methods for MCDM; The Promcalc, Gaia And Bankadviser Software**. In CarlosA. Bana e Costa, editor, *Readings in Multiple Criteria Decision Aid*, pages 216–252. Springer Berlin Heidelberg, 1990. 66

[93] T.L. Saaty. *Decision Making for Leaders: The Analytic Hierarchy Process for Decisions in a Complex World*. Analytic hierarchy process series. RWS Publications, 1990. 66

[94] H.M. Alabool and AK. Mahmood. **Review on cloud service evaluation and selection methods**. In *Research and Innovation in Information Systems (ICRIIS), 2013 International Conference on*, pages 61–66, Nov 2013. 66

[95] L. Benyoucef, H. Ding, and X. Xie. **Supplier selection problem : selection criteria and methods**. Technical Report 2003-02, INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE (INRIA), McMaster University, Hamilton, Canada, 2003. 66

[96] J.S. Dyer. **Maut — Multiattribute Utility Theory**. In *Multiple Criteria Decision Analysis: State of the Art Surveys*, **78** of *International Series in Operations Research & Management Science*, pages 265–292. Springer New York, 2005. 66

[97] A. Ishizaka and P. Nemery. *Multi-attribute utility theory*, pages 81–113. John Wiley & Sons Ltd, 2013. 66

[98] S.K. Garg, S. Versteeg, and R. Buyya. **A framework for ranking of cloud computing services**. *Future Generation Computer Systems*, **29**(4):1012 – 1023, 2013. Special Section: Utility and Cloud Computing. 67

[99] Saul I. Gass. **Model World: The Great Debate-MAUT Versus AHP**. *Interfaces*, **35**(4):308–312, July 2005. 67

[100] R.D. Luce and H. Raiffa. *Games and Decisions*. John Wiley and Sons, New York, 1957. 67

[101] W. Furlong, D. Feeny, G. Torrance, C. Goldsmith, S. DePauw, Z. Zhu, M. Denton, and M. Boyle. **Multiplicative Multi-Attribute Utility Function for the Health Utilities Index 3 (HUI3) System: A Technical Report**. Centre for Health Economics and Policy Analysis Working Paper Series 1998-11, Centre for Health Economics and Policy Analysis (CHEPA), McMaster University, Hamilton, Canada, 1998. 68

[102] Z. Wang, S. Zhang, and J. Kuang. **A Dynamic MAUT Decision Model for R&D Project Selection**. In *Computing, Control and Industrial Engineering (CCIE), 2010 International Conference on*, **1**, pages 423–427, June 2010. 69, 93

[103] M. Sun, T. Zang, X. Xu, and R. Wang. **Consumer-Centered Cloud Services Selection Using AHP**. In *Service Sciences (ICSS), 2013 International Conference on*, pages 1–6, April 2013. 69

[104] M.N. Bennani and D. Menasce. **Resource Allocation for Autonomic Data Centers using Analytic Performance Models**. In *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, pages 229–240, June 2005. 70

[105] Daniel A. Menascé and P. Ngo. **Understanding Cloud Computing: Experimentation and Capacity Planning**. 70

[106] System Reliability Center. **Quantitative Measures of Availability**. `http://src.alionscience.com/pdf/QuantitativeMeasuresAvailability.pdf`. 71

[107] K. Nomura, K. Yamori, E. Takahashi, T. Miyoshi, and Y. Tanaka. **Waiting Time versus Utility to Download Images**. In *in 2001 Asia Pacific Symposium on Information and Telecommunication Technologies*, 2009. 71

[108] H.A. Nguyen, T. Van Nguyen, and Deokjai Choi. **How to Maximize User Satisfaction Degree in Multi-service IP Networks**. In *Intelligent Information and Database Systems, 2009. ACIIDS 2009. First Asian Conference on*, pages 471–476, April 2009. 71

[109] J.Z. Wang, P. Varman, and C.S. Xie. **Optimizing storage performance in public cloud platforms**. *Journal of Zhejiang University SCIENCE C*, **12**(12):951–964, 2011. 73

[110] J. Weinman. **Time is Money: The Value of "On-Demand"**. `http://www.joeweinman.com/Resources/Joe_Weinman_Time_Is_Money.pdf`. 77

[111] S. Islam, K. Lee, A. Fekete, and A. Liu. **How a Consumer Can Measure Elasticity for Cloud Platforms**. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, ICPE '12, pages 85–96, New York, NY, USA, 2012. ACM. 77

[112] T. Limin, H. Decai, and H. Libin. **Research on Subjective Trust Model Based on Cloud Model for Open Networks**. *Journal of Computational Information Systems*, pages 4844–4854, 2011. 82

[113] S. Wang, L. Zhang, N. Ma, and S. Wang. **An Evaluation Approach of Subjective Trust Based on Cloud Model**. In *Computer Science and Software Engineering, 2008 International Conference on*, **3**, pages 1062–1068, Dec 2008. 82

[114] Z. Zhao-xiong, He XU, and W. Suo-ping. **A Novel Weighted Trust Model based on Cloud**. *Advances in Information Sciences and Service Sciences*, **3**(3), April 2011. 83

[115] S. Ries, S. Habib, M. Mühlhäuser, and V. Varadharajan. **CertainLogic: A Logic for Modeling Trust and Uncertainty**. In JonathanM. McCune, Boris Balacheff, Adrian Perrig, Ahmad-Reza Sadeghi, Angela Sasse, and Yolanta Beres, editors, *Trust and Trustworthy Computing*, **6740** of *Lecture Notes in Computer Science*, pages 254–261. Springer Berlin Heidelberg, 2011. 83, 84, 100

[116] S.M. Habib, S. Ries, and M. Muhlhauser. **Towards a Trust Management System for Cloud Computing**. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*, pages 933–939, Nov 2011. 83, 100

[117] **10 Questions to Ask When Choosing a Cloud Provider**. `http://www.entrepreneur.com/article/226845`. 99

[118] O.H. Alhazmi and Y.K. Malaiya. **Evaluating disaster recovery plans using the cloud**. In *Reliability and Maintainability Symposium (RAMS), 2013 Proceedings - Annual*, pages 1–6, Jan 2013. 100

[119] T. Wood, E. Cecchet, K. K. Ramakrishnan, P. Shenoy, J. Van Der Merwe, and A. Venkataramani. **Disaster Recovery as a Cloud Service: Economic Benefits & Deployment Challenges**. 100

[120] S. Yangui and S. Tata. **CloudServ: PaaS Resources Provisioning for Service-Based Applications**. *2014 IEEE 28th International Conference on Advanced Information Networking and Applications*, **0**:522–529, 2013. 105

[121] M. Mohamed, D. Belaid, and S. Tata. **Monitoring and Reconfiguration for OCCI Resources**. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, **1**, pages 539–546, Dec 2013. 105

[122] H. Pervaiz. **A Multi-Criteria Decision Making (MCDM) network selection model providing enhanced QoS differentiation to customers**. In *Multimedia Computing and Information Technology (MCIT), 2010 International Conference on*, pages 49–52, March 2010. 106

[123] C. Bana e Costa, J.M. De Corte, and J.C. Vansnick. **On the Mathematical Foundation of MACBETH**. In *Multiple Criteria Decision Analysis: State of the Art Surveys*, **78** of *International Series in Operations Research & Management Science*, pages 409–437. Springer New York, 2005. 106

[124] C. Bana E Costa and J.C. Vansnick. **The MACBETH Approach: Basic Ideas, Software, and an Application**. In Nadine Meskens and Marc Roubens, editors, *Advances in Decision Analysis*, **4** of *Mathematical Modelling: Theory and Applications*, pages 131–157. Springer Netherlands, 1999. 106