# AVERTISSEMENT

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.

Il est soumis à la propriété intellectuelle de l'auteur : ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document.

D'autre part, toute contrefaçon, plagiat, reproduction illicite de ce travail expose à des poursuites pénales.

Contact : portail-publi@ut-capitole.fr

# LIENS

Code la Propriété Intellectuelle – Articles L. 122-4 et L. 335-1 à L. 335-10
Loi n°92-597 du 1er juillet 1992, publiée au *Journal Officiel* du 2 juillet 1992
http://www.cfcopies.com/V2/leg/leg-droi.php
http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm


# THÈSE

ÉCOLE NATIONALE DES SCIENCES DE L'INFORMATIQUE

**En vue de l'obtention du**

## DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE ET DE L'UNIVERSITE DE MANOUBA

**Délivré par :** *l'Université Toulouse 1 Capitole (UT1 Capitole)*
Cotutelle internationale *Ecole Nationale des Sciences de l'Informatique (ENSI)*

**Présentée et soutenue le** *30/11/2017* **par :**

**Aya OMEZZINE**

## Automated and dynamic multi-level negotiation framework applied to an efficient cloud provisioning

**JURY**

| | | |
|---|---|---|
| Djamel BEN SLIMANE | Prof. à Université Lyon 1 | Rapporteur |
| Ahmed HADJ KACEM | Prof. à Université de Sfax | Rapporteur |
| Henda BEN GHEZALA | Prof. à l'ENSI Manouba | Examinateur |
| Gene COOPERMAN | Prof. à Northeastern University | Examinateur |
| Khalil DRIRA | DR au LAAS-CNRS | Examinateur |
| Muhammad Younas | DR à Oxford Brookes University | Examinateur |
| Narjès BELLAMINE BEN SAOUD | Prof. à l'ENSI Manouba | Directeur de thèse |
| Said TAZI | MCF-HDR à UT1 Capitole | Directeur de thèse |

**École doctorale et spécialité :**
*MITT : Domaine STIC : Réseaux, Télécoms, Systèmes et Architecture*
**Unité de Recherche :**
*LAAS - CNRS (UPR 8001) et RIADI*
**Directeur(s) de Thèse :**
*Narjès BELLAMINE BEN SAOUD* et *Said TAZI*
**Rapporteurs :**
*Djamel BEN SLIMANE* et *Ahmed HADJ KACEM*

# *Abstract*

Cloud provisioning is the process of deployment and management of applications on public cloud infrastructures. Cloud provisioning is used increasingly because it enables business providers to focus on their business without having to manage and invest in infrastructure. Cloud provisioning includes two levels of interaction: (1) between end-users and business providers for application provisioning; and (2) between business providers and resource providers for virtual resource provisioning. The cloud market nowadays is a complex environment where business providers need to maximize their monetary profit, and where end-users look for the most efficient services with the lowest prices. With the growth of competition in the cloud, business providers must ensure efficient provisioning that maximizes customer satisfaction and optimizes the providers' profit. So, both providers and users must be satisfied in spite of their conflicting needs. Negotiation is an appealing solution to solve conflicts and bridge the gap between providers' capabilities and users' requirements. Intuitively, automated Service Level Agreement (SLA) negotiation helps in reaching an agreement that satisfies both parties. However, to be efficient, automated negotiation should consider the properties of cloud provisioning mainly the two interaction levels, and complexities related to dynamicity (e.g., dynamically-changing resource availability, dynamic pricing, dynamic market factors related to offers and demands), which greatly impact the success of the negotiation.

The main contributions of this thesis tackling the challenge of multi-level negotiation in a dynamic context are as follows: (1) We propose a generic negotiator model that considers the dynamic nature of cloud provisioning and its potential impact on the decision-making outcome. Then, we build a multi-layer negotiation framework built upon that model by instantiating it among Cloud layers. The framework includes negotiator agents. These agents are in communication with the provisioning modules that have an impact on the quality and the price of the service to be provisioned (e.g, the scheduler, the monitor, the market prospector). (2) We propose a bilateral negotiation approach between end-users and business providers extending an existing provisioning approach. The proposed decision-making strategies for negotiation are based on communication with the provisioning modules (the scheduler and the VM provisioner) in order to optimize the business provider's profit and maximize customer satisfaction. (3) In order to maximize the number of clients, we propose an adaptive and concurrent negotiation approach as an extension of the bilateral negotiation. We propose to harness the workload changes in terms of resource availability and pricing in order to renegotiate simultaneously with multiple non-accepted users (i.e., rejected during the first negotiation session) before the establishment of the SLA. (4) In order to handle any potential SLA violation, we propose a proactive renegotiation approach after SLA establishment. The renegotiation is launched upon detecting an unexpected event (e.g., resource failure) during the provisioning process. The proposed renegotiation decision-making strategies aim to minimize the loss in profit for the provider and to ensure the continuity of the service for the consumer.

The proposed approaches are implemented and experiments prove the benefits of adding (re)negotiation to the provisioning process. The use of (re)negotiation improves the provider's profit, the number of accepted requests, and the client's satisfaction.

# *Résumé*

L'approvisionnement du Cloud est le processus de déploiement et de gestion des applications sur les infrastructures publiques du Cloud. L'approvisionnement du Cloud est de plus en plus utilisé car il permet aux fournisseurs de services métiers de se concentrer sur leurs activités sans avoir à gérer et à investir dans l'infrastructure. L'approvisionnement Cloud comprend deux niveaux d'interaction : (1) entre les utilisateurs finaux et les fournisseurs de services pour l'approvisionnement des applications, et (2) entre les fournisseurs de services et les fournisseurs de ressources pour l'approvisionnement des ressources virtuelles. L'environnement Cloud est devenu un marché complexe où tout fournisseur veut maximiser son profit monétaire et où les utilisateurs finaux recherchent les services les plus efficaces tout en minimisant leurs coûts. Avec la croissance de la concurrence dans le Cloud, les fournisseurs de services métiers doivent assurer un approvisionnement efficace qui maximise la satisfaction de la clientèle et optimise leurs profits. Ainsi, les fournisseurs et les utilisateurs doivent être satisfaits en dépit de leurs besoins contradictoires. La négociation est une solution prometteuse qui permet de résoudre les conflits en comblant le gap entre les capacités des fournisseurs et les besoins des utilisateurs. Intuitivement, la négociation automatique des contrats (SLA) permet d'aboutir à un compromis qui satisfait les deux parties. Cependant, pour être efficace, la négociation automatique doit considérer les propriétés de l'approvisionnement du Cloud et les complexités liées à la dynamicité (dynamicité de la disponibilité des ressources, dynamicité des prix). En fait ces critères ont un impact important sur le succès de la négociation. Les principales contributions de cette thèse répondant au défi de la négociation multi-niveau dans un contexte dynamique sont les suivantes: (1) Nous proposons un modèle de négociateur générique qui considère la nature dynamique de l'approvisionnement du Cloud et son impact potentiel sur les résultats décisionnels. Ensuite, nous construisons un cadre de négociation multicouche fondé sur ce modèle en l'instanciant entre les couches du Cloud. Le cadre comprend des agents négociateurs en communication avec les modules en relation avec la qualité et le prix du service à fournir (le planificateur, le moniteur, le prospecteur de marché). (2) Nous proposons une approche de négociation bilatérale entre les utilisateurs finaux et les fournisseurs de service basée sur une approche d'approvisionnement existante. Les stratégies de négociation sont basées sur la communication avec les modules d'approvisionnement (le planificateur et l'approvisionneur de machines virtuelles) afin d'optimiser les bénéfices du fournisseur de service et de maximiser la satisfaction du client. (3) Afin de maximiser le nombre de clients, nous proposons une approche de négociation adaptative et simultanée comme extension de la négociation bilatérale. Nous proposons d'exploiter les changements de charge de travail en termes de disponibilité et de tarification des ressources afin de renégocier simultanément avec plusieurs utilisateurs non acceptés (c'est-à-dire rejetés lors de la première session de négociation) avant la création du contrat SLA. (4) Afin de gérer toute violation possible de SLA, nous proposons une approche proactive de renégociation après l'établissement de SLA. La renégociation est lancée lors de la détection d'un événement inattendu (par exemple, une panne de ressources) pendant le processus d'approvisionnement. Les stratégies de renégociation proposées visent à minimiser la perte de profit pour le fournisseur et à assurer la continuité du service pour le consommateur. Les approches proposées sont mises en œuvre et les expériences prouvent les avantages d'ajouter la (re)négociation au processus d'approvisionnement. L'utilisation de la (re)négociation améliore le bénéfice du fournisseur, le nombre de demandes acceptées et la satisfaction du client.

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# Abbreviations

| | |
|---|---|
| **SaaS** | **S**oftware **A**s **A** **S**ervice |
| **PaaS** | **P**latform **A**s **A** **S**ervice |
| **IaaS** | **I**nfrastructure **A**s **A** **S**ervice |
| **MAS** | **M**ulti **A**gent **S**ystem |
| **SLA** | **S**ervice **L**evel **A**greement |
| **VM** | **V**irtual **M**achine |
| **QoS** | **Q**uality **O**f **S**ervice |
| **SOC** | **S**ervice **O**riented **A**rchitecture |
| **BTU** | **B**illing **T**ime **U**nit |

# Introduction

## 1 Context

Cloud computing presents a highly dynamic marketplace for delivering IT services on demand [6]. The cloud can be considered as a three-tier or three-layered market [1] composed of: (i) end-users; (ii) business service providers offering Software-as-a-Service (SaaS) and Platform-as-a-Service (PaaS) applications; and (iii) resource providers maintaining the physical servers and offering Infrastructure-as-a-Service (IaaS) virtual resources.

*Cloud provisioning* is the process of deployment and management of applications on public cloud infrastructures (i.e., IaaS such as Amazon EC2) [7, 8]. To run their applications, business providers may prefer to rent virtual resources from IaaS providers instead of in-house hosting. By doing so, they avoid infrastructure maintenance and can scale their application to serve as many end-users as possible. As business fluctuates, the business providers can also scale down. Renting rather than owning resources enables business providers to focus on their business, without having to manage and invest in infrastructure. In addition, they benefit not only from the pay-per-use model but also from the competition between different resource providers offering attractive offers. Cloud provisioning is composed of two interaction levels. The first one is between end-users and business provider(s) for application provisioning, while the second deals with Virtual Machine (VM) provisioning between the business provider and the resource provider(s). In this thesis, we focus on the business provider, who is responsible for application and VMs provisioning.

Cloud provisioning is generally governed by a *Service Level Agreement* (SLA) contract established between the parties concerned (e.g., user and business provider, business provider and IaaS provider). An SLA is a formal representation of the *Quality Of Service* (QoS), the penalties and the obligations agreed upon by the contractors [9]. For efficient cloud provisioning, business providers aim to optimize their monetary profit and to satisfy as many end-users' requests as possible. To do so, business providers require SLA- and profit-aware provisioning strategies. In other words, the business provider should guarantee that: (i) the SLA is met and (ii) it is also making some profit by serving end-users.

Despite its advantages, cloud provisioning has some complexities due essentially to the dynamic nature of the cloud environment (e.g., dynamically-changing resource availability, dynamic pricing, and dynamic market factors related to offers and demands) [7]. Efficient cloud provisioning that satisfies the end-users and maximizes the providers' profit is not a trivial task when considering the dynamicity of the cloud. Among the limitations impacting the efficiency of cloud provisioning is the inflexibility of SLAs. Indeed, contemporary providers bind their services to inflexible take-it-or-leave-it SLAs [10]. Alternatively, some proposed approaches use admission control strategies able to evaluate the request according to the provider's capabilities and such approaches decide whether to accept or reject the request. When it is not possible to provide the requested service (while satisfying the obligations of the SLA), the provider rejects the request in order to avoid an SLA violation. By considering the constraints of resource provisioning (e.g., cost and availability), the business provider may lose several clients, which may lead to a loss of potential profit. This is due to the gap between clients' requirements and business providers capabilities. In fact, each cloud actor (e.g., the end-user, business provider, and resource provider) has its own interests. In particular, the client aims to obtain the most convenient service at the cheapest price, while the service/resource provider aims to maximize its profit and to serve the maximum number of clients. The negotiation between cloud actors is an intuitive way to solve conflicts between clients and providers and to reach a satisfactory agreement. As the infrastructures and platforms for the cloud became more complex, it created the need for automated negotiation to handle dynamic and multi-level concurrent interactions.

Automated negotiation is a promising solution when dealing with conflicts and brings flexibility to the SLA establishment process. It is considered as an important step in the SLA life-cycle [2]. Automated negotiation has three primary aspects [11]: (i) the negotiation protocol, which defines the rules of interaction; (ii) the negotiated service, which is composed of objects (also called issues) about which the participants negotiate; and (iii) the decision-making model, which defines the decision-making strategies for each actor.

Intuitively, negotiation helps in reaching an agreement that satisfies both parties. However, in order to be efficient, the negotiation should consider the cloud provisioning properties (e.g., two interaction levels) and complexities related to dynamicity, which greatly impacts the success of the negotiation. In fact, according to the state-of-the-art, cloud provisioning characteristics are insufficiently taken into account by the current negotiation approaches.

The main focus of this thesis is to deal with negotiation for an efficient cloud provisioning by proposing a conceptual negotiation framework suitable for cloud provisioning. Also, we propose three negotiation approaches related to SaaS application provisioning and especially to compute-intensive applications. In fact, the cloud is evolving to include a High Performance Computing (HPC) environment that is able to perform large-scale and resource-intensive applications by offering on-demand computing power [12].

## 2 Thesis research issue and challenges

Our thesis research issue is : **How to automate multi-level negotiation for an efficient cloud provisioning that satisfies end-users and optimizes business provider profit in a dynamic context ?**

Cloud provisioning has several properties such as the two interaction provisioning levels and the dynamicity at both business side (e.g., dynamic pricing, variation in market offers) and resource side (e.g., variation in resource availability). Cloud provisioning properties on one hand, and the business provider's objectives on the other hand make an efficient negotiation a challenging task. Despite active research in the context of cloud negotiation, some issues are still not addressed (e.g., multi-level, dynamicity). The research issue question raises four challenges. The first challenge is related to negotiation for cloud provisioning in general. The others deal with issues related to an efficient SaaS application provisioning:

### Question 1: How the negotiation should be designed for an efficient cloud provisioning ?

Cloud provisioning has two main properties that highly influence the negotiation outcome and the way to achieve an agreement that satisfies both parties. The first one is related to the multi-layer aspect of the cloud. Each cloud layer offers one type of service, which may depend on another layer. In fact, the application provisioning depends on the resource provisioning from the IaaS layer. Consequently, the negotiation with end-users for application provisioning will depend on resource provisioning. The second one is related to the *dynamic context of provisioning*, where the context refers to the elements in relation with the quality of the negotiated service and its price. In fact, in addition to negotiators' preferences which may be variable, there may be several dynamic elements that could influence the negotiation process. Those elements may be related to the provisioning process itself such as resource allocation, monitoring, or some other elements related to dynamic market status.

Despite their importance, the two properties mentioned above are not well addressed in the current negotiation models. In fact, concerning the multi-layer aspect, most of the literature focuses on negotiation in a specific layer and primarily in resource negotiation at the IaaS layer [13]. The negotiation at the business layer is not well addressed and the elaborated approaches [10, 14, 15] do not consider the dynamic resource provisioning. Some existing work addresses the multi-layer aspect [10, 15, 16], but do not consider the decision-making strategies across provisioning levels. Furthermore, concerning the second property (dynamic context of provisioning), most of the literature proposes a decision-making strategy that considers fixed elements. Thus, those strategies are appropriate for a given negotiation situation (e.g., [17, 18]). Given the cloud

dynamicity, when the situation changes, the strategy may not be relevant. In addition, the majority of the elaborated approaches assumes that the negotiation is an independent activity of the provisioning process (i.e., independent of the allocation and the monitoring activities) [14, 19].

In conclusion, there are missing features (multi-layer, dynamic context) in current negotiation models, that should be considered for an efficient cloud provisioning. As mentioned earlier, while this challenge is related to cloud negotiation in general, the following challenges deal with SaaS-application negotiation and especially compute-intensive application.

### Question 2: How SaaS application negotiation could maximize customer satisfaction and optimize business provider profit ?

With the growth of competitiveness in cloud, business providers must ensure an efficient provisioning which maximizes customer satisfaction and optimizes their profit. Indeed, for each incoming request, the provider must take the right decision about its placement on rented VMs, while satisfying the QoS requirements and maximizing profit. To optimize the profit, the business provider should maximize the budget given by clients, minimize the costs of rented VMs and minimize penalty costs. The majority of the elaborated approaches for application provisioning focus on costs of rented VMs. In fact, some work considers both profit optimization and customer satisfaction [1, 20]. However, as mentioned earlier, this work (as most provisioning approaches) suffers from inflexible SLAs by either using a take-it-or-leave-it strategy or admission control strategy.

The negotiation between end-users and the business provider brings flexibility to the provisioning process, by trying to find a compromise satisfying both parties. But, for an efficient provisioning, the negotiation decision-making strategies should consider elements related to the provisioning process such as request scheduling and VM provisioning strategies. In fact, an efficient assignment of application requests on leased VMs impacts highly on customer satisfaction and business provider's profit.

In contrast to IaaS negotiation, SaaS application negotiation is not well developed in the literature [13]. Furthermore, the current SaaS negotiation models do not consider scheduling and VM provisioning in their decision-making strategies. This work leads to an inefficient profit optimization since they do not consider the prices of the resources needed by the user's request. Also, the customer satisfaction is not guaranteed because the performance of the application depends highly on the VMs executing it.

So, the second challenge consists in proposing a bilateral negotiation approaches between the end-user and the business provider aiming to optimize the business provider's profit and maximizing the customer satisfaction.

### Question 3: How the business provider could maximize the number of clients given the constraints related to resource availability and pricing at negotiation time ?

In order to maximize the number of clients, the business provider should be able not only to accept as many incoming requests as possible but also to guarantee the clients' requirements. This is a challenging task when considering finite resources at a time and concurrency among *heterogeneous clients' requests* (i.e., having different requirements with different budget). In fact, reaching an agreement is not usually feasible due to the constraints related to resources availability and pricing at negotiation time. Also, the clients may have specific requirements with limited budget that the provider could not satisfy when considering resources status and prices at negotiation time. In addition, negotiation sessions are generally limited by time (which is called negotiation deadline). If the two parties do not reach an agreement before the negotiation deadline, the negotiation ends up with failure.

The majority of the elaborated negotiation approaches are mono-session (e.g., [19, 21, 22]). Those approaches enable the exchange of offers and counter offers (i.e., multi round) only during the negotiation session which is limited by the negotiation deadline. This solution may lead to an increased number of rejected users. This is because, the negotiation decision-making strategies are based on the current situation constraints (e.g., insufficient resources and high resources' prices) that may not allow for an agreement to be reached.

In addition to the mono-session aspect, the concurrency among clients' requests are insufficiently taken into consideration by the current negotiation approaches.

In order to minimize the number of rejected users' requests, the business provider should use an adaptive negotiation approach that considers the workload change (resource availability and pricing) and keeps negotiating concurrently with all users according to those changes. So, the provider can accept more users which will lead to an increased profit and better satisfaction of end-users.

So, the third challenge consists of proposing an adaptive negotiation mechanism according to resource availability and pricing variation in order to maximize the number of clients.

### Question 4: How to overcome the problem of unexpected events leading to potential SLA violation ?

Cloud computing represents a highly dynamic environment (both at the business level and at the resource level). There may be unforeseen events at the resource level such as catastrophic resource failure, or there may be unexpected events at the business level coming from the need to share rented resources between new clients that compete for immediate execution. These events

may result in a violation of the original negotiated SLA since the schedule originally done (based on the initial SLA) can be modified.

Generally if a contract is violated, a penalty is paid and the service is canceled [23, 24]. But if a contract is violated due to circumstances not accounted for in the original SLA negotiation, the two parties may *both* lose badly. For example, consider the situation in which a job is critical to the success of the business. In principle, the client could have insisted on a penalty in the original SLA that is equal to the value of the client's business, as a compensation for the losses resulting from the failure of that business-critical job. But this is usually unrealistic, since such a penalty could be even larger than the total assets of the provider. Hence, the client will never be fully compensated, and the provider faces a loss of future clients due to the loss in reputation as the number of violated jobs accumulates. For these reasons, the provider and the client would normally prefer to renegotiate a new SLA with new issues' values rather than pay a steep penalty and accept the cancellation of a business-critical job.

Most of the literature assumes that once an SLA is established, it cannot be renegotiated [23–26]. The concept of *renegotiation* has not yet been well studied [27]. The idea of renegotiation had been invoked by some works, whereas there is no a concrete decision-making strategy (e.g., [13, 28]) or the renegotiation decision-making is partially considered [29]. There is some work that tries to enhance the WS-Agreement negotiation protocol using renegotiation [29–31], and others propose general conceptual renegotiation frameworks [32, 33]. The work mentioned above do not propose any decision-making model to guide the renegotiation process toward a satisfactory agreement.

So, the fourth challenge consist in proposing a renegotiation approach aiming to ensure the continuity of service and minimize the loss in profit due to potential SLA violation.

# 3 Research Aims and Contributions

The aim is to propose a novel negotiation framework to handle the issues presented previously and deal with the limitations of existing approaches. The contributions of the thesis can be summarized as follows:

### Contribution 1: Negotiation-based framework applied to cloud provisioning

Our objective is to propose a negotiation framework taking into account cloud provisioning properties (multi-layer and dynamicity).

First, we propose a generic negotiation-based provisioning process which contains the most important activities to be considered for an efficient negotiation-based provisioning in the cloud.

The proposed process shows how the negotiation could be guided by the provisioning activities, especially allocation and monitoring. Also, it details the different cases where it is important to renegotiate either before SLA establishment or after SLA establishment.
Second, we focus on the (re)negotiation activity as part of the overall provisioning process by presenting the negotiator model and the behavior process of the negotiator during the (re)negotiation phase. The negotiator model can be instantiated in each cloud layer as an agent which is able to act on behalf of any cloud actor (user, business provider, resource provider). The negotiator's decision-making strategy takes into account dynamic provisioning context which consists of the elements impacting the QoS of the negotiated service and its price. The decision-making strategy is designed in a way to consider potential dynamic changes and to act accordingly.
Finally and based on the negotiator model and its instantiation among cloud layers, we propose a multi-layer negotiation framework that considers the dynamic nature of cloud provisioning and its potential impact on the decision-making outcome.

### Contribution 2: Bilateral negotiation for an efficient SaaS application provisioning

In order to maximize business provider's profit and increase customer satisfaction, we propose a negotiation approach between end-users and the business provider based on an existing provisioning approach. That provisioning approach is composed of two phases: (i) Admission control phase which evaluates and decides whether to accept or reject a request based on business provider's capabilities. (ii) Scheduling phase which is responsible for request assignment to virtual resources.
First, we propose to extend the classical admission control in order to support negotiation by storing the reasons of rejection. Then, we integrate the negotiation capabilities to the overall provisioning process. The core idea of our solution is, when it is not possible to schedule a request respecting the initial SLA, the business provider may propose other scheduling alternatives instead of rejecting the request. The scheduling alternatives are generated through the communication with the provisioning modules such as admission control, scheduler and VM provisioner. We propose two decision-making strategies based on the generated scheduling alternatives. The business provider follows a decision-making strategy according to his goal.
The experiments show the benefits of adding negotiation to the provisioning process by improving the business provider's profit, the number of accepted users' requests, and the client's satisfaction.

### Contribution 3: Adaptive and concurrent negotiation for an efficient SaaS application provisioning

In order to maximize the number of clients, we propose to extend the bilateral negotiation approach previously described to deal with the negotiation failure cases at point of time $t$. Given the dynamicity of the cloud (variation of resource availability, dynamic resource pricing, etc.), at point of time $t + 1$, there may be new elements such as new idle resource and new resource capacity which may change the previous negotiation outcome. We propose to harness those changes by adapting the negotiation behavior accordingly. In fact, in dynamic environment it is important to use a negotiation approach aware of the system changes. We propose an adaptive negotiation approach that considers the workload change and keeps negotiating concurrently with non-accepted users according to those changes. So, the business provider can accept more users which will lead to an increased profit and better satisfaction of end-users. To do so, when the negotiation fails, the business provider gives other chances to non-accepted users by renegotiating (before SLA establishment) simultaneously with them when there is a change in the elements considered in the first negotiation. Those elements change are given by the business provider's provisioning modules (VM provisioner and scheduler). The experiments show that the adaptive and concurrent negotiation improves the business provider's profit, the number of accepted users' requests, and the client's satisfaction, as compared with the bilateral negotiation.

### Contribution 4: Proactive renegotiation for an efficient SaaS provisioning

In order to convert the lose-lose situation due to SLA violation into a win-win one, we propose a proactive renegotiation approach after SLA establishment. When an event threatens a lose-lose violation of the SLA, the renegotiation is launched to establish a new SLA that limits the losses on the two sides. We propose an automated renegotiation-based approach when detecting an unexpected event during the SaaS provisioning process. In our approach, the provider proactively renegotiates with the clients whose jobs may be in violation of the SLAs, in order to minimize the loss in profit and in order to ensure the continuity of the service. The renegotiation approach is composed of two steps.

(i) The first step happens when the business provider detects an unexpected event. Since the provider may not be able to physically continue some jobs with the same scheduling parameters (VM, completion time, etc.), we consider alternative rescheduling options for the business provider. The first step consists of the selection of an option for profit-aware rescheduling. In examining the possible scheduling options, the provider chooses an option leading to a minimum loss in profit while also minimizing the number of canceled jobs.

(ii) At the second step, the business provider triggers a renegotiation with those end-users whose

jobs may terminate after deadline. The strategies followed by the business provider are based on the rescheduling option selected in the first step.
The experiments show that this new approach minimizes the loss in profit of the business provider and minimizes the number of cancelled jobs, as compared with enforcing the original SLA.

## 4 Structure of the Thesis

The rest of this thesis is organized as follows. In Chapter I, we present basic concepts related to our research context (e.g., cloud computing, automated negotiation). In Chapter II, we review related literature on cloud provisioning and SLA negotiation. In Chapter III, we present our multi-layer and dynamic negotiation framework applied to cloud provisioning. In Chapter 4, we propose a bilateral negotiation approach between end-users and a business provider aiming to maximize customer satisfaction and optimize the provider's profit. The negotiation decision-making strategies are based on the communication with the provisioning modules (e.g., the scheduler and the VM provisioner). In Chapter V, the bilateral negotiation is extended in order to support concurrent and adaptive negotiation according to workload variation in terms of resource availability and pricing. When the bilateral negotiation fails, the business provider renegotiates simultaneously with those users when there is a change in the elements considered in the first negotiation. In Chapter VI, we propose a renegotiation approach after SLA establishment between end-users and a business provider aiming to minimize SLA penalty costs and to ensure the continuity of the service. Finally, we conclude the thesis and draw some future directions.

# Chapter I

# Basic concepts

Contents

# 1 Introduction

The cloud computing is increasingly used for IT service delivery. The cloud is evolving towards a highly dynamic market where each actor has its own interest. Automated negotiation is a compelling way to reach a satisfactory agreement between two or more parties with conflicting needs. It is generally adopted in the Cloud in order to enable negotiators to establish a common agreement, written as a Service Level agreement (SLA) contract. In Section 2, we provide a brief overview about cloud computing and cloud provisioning. In Section 3, we present the main concepts of automated negotiation. In Section 4, we give a description of SLA life-cycle and SLA-based cloud provisioning process. Finally, we conclude the chapter in Section 5.

# 2 Cloud computing

## 2.1 Cloud definition

Several definitions had been proposed for cloud computing. According to National Institute of Standards and Technology (NIST) [34], the cloud computing is *"a model for enabling ubiquitous, 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"*. In [35], Furht defines the cloud computing as *"a new style of computing in which dynamically scalable and often virtualized resources are provided as a services over the Internet."*
In [9], Buyya et al. propose the following definition including SLA negotiation:
*"A Cloud is a type of parallel and distributed system consisting of a collection of inter-connected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resource(s) based on service-level agreements established through negotiation between the service provider and consumers."*

## 2.2 Cloud characteristics

According to NIST, the cloud has five essential characteristics:

- On-demand self-service "X as service": cloud resources (e.g., infrastructure, platform, application) are viewed as a services and can be provisioned automatically without human interaction.
- Broad network access: cloud resources are available over the network and can be accessed by heterogeneous thin or thick client platforms (e.g., mobile phones, laptops).

- Resource sharing: cloud resources are virtualized and shared among multiple consumers using the multi-tenant model. The consumers have no information about the structure hidden behind the resource provisioned.
- Elasticity and scalability: the resource capabilities can be quickly increased (scale up) or released (scale down) according to the demand.
- Measured service: cloud services can be measured, monitored and controlled with a transparent manner for both the provider and the consumer.

## 2.3 Cloud services

The cloud offers three type of service models:

- Software as a Service (SaaS): the capability to offer applications running on cloud infrastructure. The consumers can access to the applications from everywhere via the internet without any control on hardware requirements. There are two main type of SaaS applications: (i) enterprise-based applications, such as Customer Relationship Management (CRM) solutions (e.g., salesforce[1]); or (ii) compute-intensive applications, such scientific data processing application.
- Platform as a Service (PaaS): the capability to offer platform including programming tools and languages to create and deploy application onto cloud infrastructure. The consumers have control on the deployed applications without any control on the cloud infrastructure. Among PaaS solutions, we can cite Microsoft Azure[2] and Google App Engine[3].
- Infrastructure as a Service (IaaS): the capability to offer computing resources (e.g., processing, storage, network), where the consumer can deploy any operating system and application without any control on the cloud infrastructure. Amazon Elastic Compute Cloud (EC2)[4] is an example of an IaaS solution.

The above mentioned cloud services can be presented as a layered cloud computing architecture, where each layer is associated to one type of service.

## 2.4 Cloud market

The cloud computing is evolving toward an open marketplace [6]. The cloud can be seen as a three-tier cloud market [1]. The Figure I.1 shows the different actors in the cloud market.

[1] https://www.salesforce.com/
[2] https://azure.microsoft.com/fr-fr/
[3] https://cloud.google.com/appengine/?hl=fr
[4] https://aws.amazon.com/fr/ec2/

The business providers (or service providers) offer both SaaS and PaaS services while resource providers offer IaaS services. To execute their services, business providers need to rent VM instances from resource providers. So, resource providers charge business providers for renting virtual resource while the business providers charge customers for processing requests. The main goal of a business provider is to maximize his/her profit by maximizing the revenue and minimizing the costs of rented VMs.

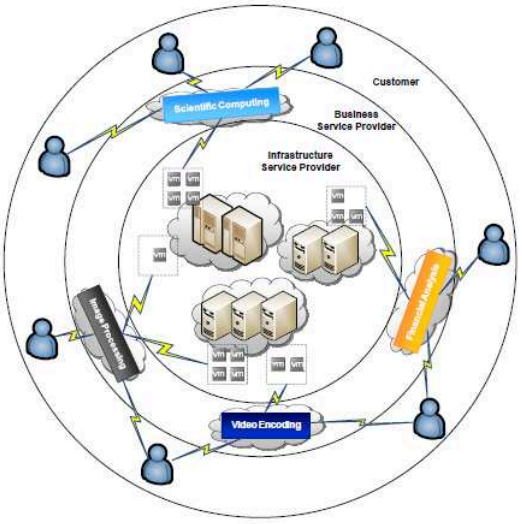


FIGURE I.1: Three-tier cloud computing market [1]

## 2.5 Cloud provisioning definition

The cloud provisioning is the process of deployment and management of applications on public cloud infrastructures (i.e., IaaS such as Amazon EC2) [7, 8]. The cloud provisioning has three steps:

- **VM provisioning** which is responsible for VM(s) instantiation. The VM(s) instantiated should match application requirements in terms of hardware and software. Actually, cloud resource providers offer different types of VMs with different characteristics and different prices. For example, Amazon EC2 proposes three billing models for VMs: (i) On-demand model enables the consumer to start using the instances immediately [21]. The consumer pays per Billing Time Unit (BTU), for Amazon EC2 the unit is equal to one hour. (ii) Reserved model requires consumers to pay for one- or three-year term but benefit from a discount on price per hour compared with the on-demand model. (iii) Spot instances are charged dynamically according to spot prices set by Amazon. The spot prices depends on the supply and the demand for spot instances. The consumers' requests are processed if their prices are greater than the spot price. If the spot price changes, and the request's

price is below the spot price, so the consumer's application is interrupted. Consumers can benefit from saving costs by using that type of instances but it cannot be suitable for all type of application.

- **Resource provisioning** which responsible for VMs placement into physical servers (called also scheduling). The scheduling is done after mapping VMs characteristics onto physical resources ones. The most used resource allocation policies by the IaaS providers are Best Effort and Immediate Reservation [36]. For the Best Effort policy, the request contains both the *start time* and the *duration*. If there are available resources, the request is assigned to those resources else the request is placed in a FIFO queue. The Immediate reservation policy is a form of advance reservation with an immediate start (used by Eucalyptus [37]). If there are available resources, the request is immediately assigned to those resources else the request is rejected.
- **Application provisioning** includes the deployment of specialized applications within VMs. Furthermore, it includes the mapping and the scheduling of end-users' requests on application instances. The mapping consists of translating the application requirements into resource needed for its execution. In fact, the requests are received in terms of software/platform characteristics, and resources are needed to realize these requests. The mapping allows the business provider to define the amount of resource needed for each request. The scheduling defines when and where to place the accepted end-users' requests.

So, the cloud provisioning may include two levels of provisioning according to the requested service (application, resource). In fact, the service provisioning is defined as the transfer of value from the provider to the requester [2]. The first level (*level 1*) between end-users and service providers and the second one (*level 2*) between service providers and resource providers.

## 2.6 Cloud provisioning process

Based on the previous definition, we illustrate the cloud provisioning process in Figure I.2. The Figure shows the different actors involved in the cloud provisioning process: end-users (*Users layer*), business providers (*Business layer*) and resource providers (*Resource layer*). The business provider is responsible for application provisioning (application request mapping and scheduling) and VM provisioning (VM(s) instantiation). The resource provider is responsible for resource provisioning (VM request mapping and scheduling). Scheduling is the core of any resource management process. There are two types of scheduling in the cloud [24], depending on the type of resource considered, physical or virtual: (i) the scheduling on the resource provider side deals with VM placement on physical servers and finds where and when to place a VM [36]. The scheduling should ensure an efficient utilization of physical servers; (ii) the scheduling on

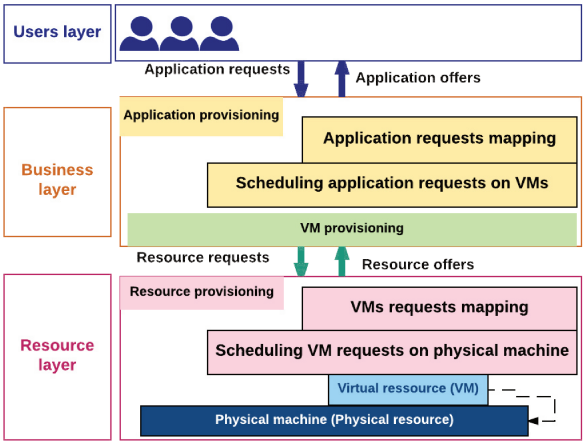


FIGURE I.2: Cloud provisioning process

the business provider side, consists in assigning end-users' requests to virtual resources that may be owned by another cloud tier. The scheduling decides in which VMs and when to place an application request. The aim of scheduling at business side is to offer a compromise between the number of resources rented and the number of accepted users [20]. For a business provider, the application provisioning should ensure an efficient virtual resource utilization and the VM provisioning should guarantee the QoS requested by end-users. This could be done via an efficient mapping and scheduling.

# 3 Automated negotiation basic concepts

## 3.1 Definition

Negotiation is the process that the different parties undertake to reach a satisfactory agreement on some matter [11]. There are two types of negotiation [38]: (i) cooperative one when negotiators aim to reach a common goal, generally the preferences are shared between negotiators. (ii) competitive one when negotiators are self-interested and each party tries to maximize its gain, preferences are kept private. In this thesis, we focus on competitive negotiation between cloud users and providers.

Negotiation has been developed beyond human interaction to involve IT interactions, for instance negotiation between software agents in an e-commerce transaction. With the evolution of infrastructures and platforms leading to highly distributed environment (e.g, cloud environment), we need automated negotiation that handles complex interactions.

Automated negotiation can be split into three topics [11]. (i) The *negotiation protocol* defines the rules of interaction between negotiators. (ii) The *negotiated service* is composed of objects (also

called issues) about which the participants negotiate. (iii) The *decision-making models* defines the decision-making strategies for each actor.

## 3.2 Negotiation protocol

The negotiation protocol includes the participants in the negotiation and can be represented by a state transition diagram containing (i) the negotiation states (ii) the legal transitions between states and (iii) the locutions that may be exchanged between the participants. There are three types of protocol, according to the number of participants in a the negotiation.

**One-to-one (bilateral):** One consumer and one provider negotiate for a specific type of service. The most commonly used protocol is the Rubinstein alternating offer protocol [39] or the FIPA contract net protocol [4]. A detailed description about those protocols is given in Appendix A.

**One-to-many:** There are two subtypes. The first one concerns a consumer who negotiates with many providers offering the same type of service. The goal of the negotiation in this case is to choose the best service at the cheapest price that still meets the consumer requirements. The second type deals with one provider negotiating with many consumers in order to have the best profit such as in an English or Dutch auction [40].

**Many-to-many:** In the case of many-to-many participants, we deal with concurrent one-to-many negotiation sessions. In this case, the protocol should define a coordination mechanism between the concurrent negotiation sessions [16, 41, 42].

## 3.3 Negotiation objects

The negotiation objects represent issues on which participants must agree. The agreement may contain one or more issue(s). The Multi-issue negotiation is more complex than the single-issue one due to the potential interdependence among them [43]. There are two types of issues:

- service-specific issues concern the negotiated service type. Those issues represent QoS parameters related to the service. In fact, each type of service has specific QoS attributes [44]. For example, for IaaS service, the negotiation could be about CPU capacity, memory size, storage, etc. For PaaS service, the negotiation could be about scalability. And for SaaS service, an agreement may contain reliability and availability.
- generic issues match all types of services, such as price and security issue.

## 3.4 Decision making models

The decision making model depends on the negotiation protocol, the nature of the negotiation issues, and the operations that can be performed on it [11]. The negotiation model enables negotiators to generate and evaluate offers during the negotiation process in order to realize their objectives. First, to evaluate an offer, the model is generally based on utility functions and negotiators' preferences. Second, to generate an offer, the model employs a strategy that determines at each step the counter offer that will be sent to the opponent.

### 3.4.1 Utility function

The utility function measures the level of satisfaction of a given offer, in other words how good is that offer [45]. The utility function is used to express customer satisfaction which measures the service performance against customer preferences and expectation [1]. In the business world, the customer satisfaction is a key indicator.
Generally, a single attribute utility function is used for the evaluation of one issue. For an offer with multiple issues, an aggregate utility function should be used. The aggregate utility function can be additive (i.e weighed sum of single attribute utility function) or non-additive mainly when there is a dependence between issues [46]. Before starting negotiation, the negotiator sets a minimum utility value under which offers are not accepted. The minimum utility is generally denoted by *reserved utility*.
For offer generation, the negotiators generally use the the *indifference curve* which represent the combination of issues leading to the same utility

### 3.4.2 Negotiation decision-making strategy

For counter offer generation, two basic strategies may be used according to the negotiators' goals:

**Concession** At each step, the negotiator decreases the total utility value by decreasing the utility of individual issues. The strategy consists in determining the amount of concession at each step using a specific pattern. Several techniques had been proposed in order to determine the concession patterns. Among the negotiation techniques, we can cite [47]: (1) Time-dependent: consider the time given for negotiation and model the fact that negotiators should concede faster as the deadline approaches. (2) Resource dependent: consider the resource remaining during negotiation. (3) Behavior-dependent: the negotiator imitates his opponent's behavior and generates counter offer(s) accordingly. (4) Market-dependent: is used for multilateral negotiation scenario and generally

considers a combination of three criteria [48]. The first one is Time as described for Time-dependent. The second criteria is the probability of finding an agreement closer to the negotiator's preferences (expressed by an *opportunity function*). The third criteria is the number of competitors and the number of available options in the market (expressed by an *competition function*) [41]. (5) Policy-based: that technique captures user preferences in the form of policies. Policies are sets of rules which define actions to be taken when certain conditions are met [49]. The policy-based technique is based on the mapping of high-level policies expressing business goals to low-level metrics defining the concession degree [50].

**Trade-off** At each step, the negotiator decreases some individual attributes' utility values and increases others in order to keep the total utility value unchanged. The degree of trade-off can be defined using the *fuzzy similarity* concept. The fuzzy similarity is based on the fact that the more similarity there is between the offer to send and the received offer, the higher the chance to reach an agreement satisfying both parties.

## 3.5 Negotiation approaches

There are three main approaches to automated negotiation in the multi-agent domain [51]:

**Game Theory** It consists of a set of players performing strategic interactions. Each player has a set of possible actions that can be performed and a function that measures the outcome for each action taken. The aim of each player is to perform actions that maximize its outcome. In game theory, the participants should have finite strategies (possible actions and their outcome). In fact, both the number of negotiators' choices and the number of rounds are finite. Also the participants should have a full knowledge of others' preferences [11]. These assumptions make a game theory approach poorly adapted for a dynamic and highly distributed environment like Cloud computing; indeed the number of participants and the number of issues may be very high. Also the strategies in such a dynamic environment cannot be finite.

**Heuristics** The heuristic approach is based on the representation of the negotiators' proposals and counter proposals as points in the space with their outcomes. The outcomes are calculated according to the negotiators' preferences and their utility functions. The possible agreements are in the intersection of the negotiators' acceptable outcomes sets. The aim is to find sub-optimal solutions [11]. Generally, negotiators are interested in *pareto optimal* offer which represents a combination of issues that cannot be better for one without being worst for the other [52].

**Argumentation** The argumentation approach is based on arguments exchanged in order to persuade others and change their beliefs [53–56]. This approach permits the sharing of additional information about negotiators' belief or other circumstances. The arguments exchanged may be used to justify a proposal, to support an offer, or to explain a fact. The argumentation is based on the interaction between arguments and how to reach a consistent conclusion [57]. The main elements of an argumentation-based negotiation are (i) argument and proposal evaluation: consider both objective (quality of proof) and subjective evaluation (preference); (ii) argument and proposal generation: it may be done using different approaches such as explicit rules (if-then) or a planning approach; and (iii) argument and proposal selection: based on the relationships between arguments. At each step the negotiator should select the most relevant argument from its sets of arguments in response to the opponent's proposal [58]. The elements cited above require a high computational and communication load, and the efficiency of argumentation depends on the application. To be pertinent as a negotiation approach, the overload must be compensated by its potential benefits [57]. The application of argumentation as a negotiation approach in Cloud computing is a new challenge [59, 60]. It may not be efficient in all cases.

# 4 Service Level Agreement (SLA)

## 4.1 Definition

In service-oriented systems (e.g., service-oriented computing, grid computing, cloud computing), both clients and provider need to control the service requested/offered. This mutual control can be expressed in a contact which specify not only the functional and non functional properties of the service, but also the price and penalties for non-compliance. In [45], the authors define the SLA as *"a contractual obligation between the service provider and the service consumer by specifying mutually-agreed understandings and expectations of the provision of a service"*.
An SLA contract can be seen as a formal representation of values for Quality of Service (QoS), obligations and penalties agreed upon by both parties (consumer and provider). For the consumer, the SLA is a guarantee to have the required service and hence motivates the use of cloud services. Given the excepted requirements of the consumers, the provider can efficiently manage its capacity.

## 4.2 SLA life-cycle

The SLA life-cycle contains the needed activities for service provisioning and SLA Management (SLAM). The SLAs are established not only to express Service level but also contain information for supporting the service provisioning activities [2]. In the literature, There are various SLA life-cycles [2, 9, 27]. Here, we present the SLA life-cycle proposed in [2] because it shows the different cases where the negotiation could happen between the contractors. As shown in figure I.3, the SLA life-cycle has six steps:

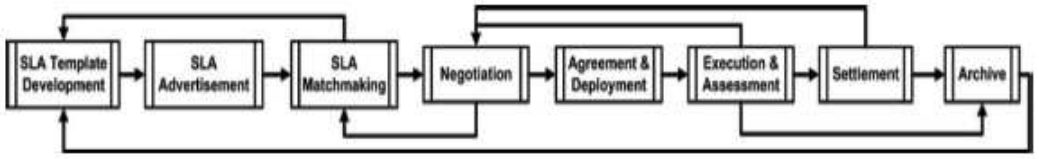


FIGURE I.3: SLA life-cycle [2]

1. Description (SLA template development and SLA advertisement): The SLA template is described according to the offered service. Then, the service template is published by the service providers.
2. Matchmaking: The consumers searches for the providers that offer services matching with their requirements.
3. Negotiation: In order to bridge the gap between the offered service and the requested one, the consumer and the selected provider may negotiate the SLA template terms.
4. Agreement and deployment: If the negotiation succeeded, the SLA is established according to the template and signed by the concerning parties.
5. Monitoring and assessment: During the service execution, the service is periodically monitored and the service level is assessed and compared to the established SLA. If there are violations, the SLA may be renegotiated or cancelled.
6. Termination (SLA settlement and SLA archive): The SLA terminates either when the contract period is over or when there is an SLA violation. In the case of SLA violation, a penalty must be paid according to the signed SLA. At this step and after service execution, the two parties may negotiate for service re-execution. Finally, the SLA is disposed or archived according to concerning parties policies.

## 4.3 SLA-based cloud provisioning process

The figure I.4 shows the activities of an SLA-based cloud provisioning process regardless of the provisioning level (application or resource). The presented process in based on the cloud provisioning definition, the SLA life-cycle and the SLAM life-cycle proposed in [27]. The SLA-based provisioning process include six steps joint to the SLA life-cycle:

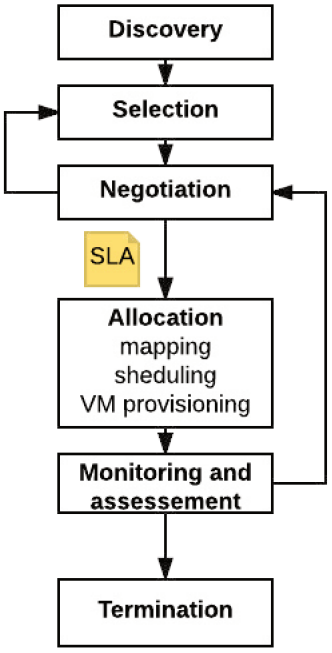


FIGURE I.4: SLA-based cloud provisioning process

1. Discovery: The first activity consists in the automatic detection of service/resource offered by the cloud providers [61]. Generally, consumers submit their requests to a broker who searches for appropriate providers [62].

2. Selection: The consumers choose the provider having the optimal offer (price and QoS). If the service is composite, the selection aims to fulfill the overall QoS.

3. Negotiation: the negotiation may be carried out after or before the selection activity. In some cases, the negotiation may assist consumers to choose the best provider(s).

4. Allocation: For the business provider the allocation step includes: (i) the application request mapping to the needed virtual resources and (ii) the scheduling of the request on the appropriate VM(s) by VM provisioning. For the resource provider, the allocation includes the mapping and the scheduling of VMs request on physical servers. The allocation should be done with a profitable manner while respecting requests' QoS requirements [61]. For

that reasons, some providers use admission control algorithm which determines which request to accept. The algorithm is executed whenever a new request arrives. It checks two conditions: (i) the feasibility condition checks if there is enough resources to execute the request. (ii) the profitability condition checks if the request acceptance brings some "monetary" profit to the provider. If the two conditions are satisfied, the request is accepted; if not, it is rejected [63]. The admission control has been used as a general mechanism to avoid overloading of resources while ensuring SLA satisfaction [20].

5. Monitoring and Assessment: In dynamic environment, it is important to continuously measure the system performance (e.g physical and virtual servers, network performance, application running on public infrastructure) [27]. The data collected from monitoring components is evaluated and it is used to predict or detect SLA violation. If it is the case, the SLA may be renegotiated.

6. Termination: The SLA based provisioning process may terminate with one of the three cases: (1) without agreement, the two parties do not reach an agreement, no SLA was established. (2) SLA established and respected without paying penalty, (3) SLA established but there is violation, in this case a penalty is paid.

# 5 Conclusion

In this chapter we presented the basic concepts related to our research context. We introduced the principles of cloud computing and cloud provisioning. Afterwards, we presented the main concepts of automated negotiation. Finally, we presented the SLA contract by describing the SLA life-cycle and proposing an SLA-based cloud provisioning process.

In the next chapter, we present a detailed description and analysis of the state-of-the-art focusing on two research areas: (i) Cloud provisioning approaches used by the business provider and (ii) negotiation approaches in utility computing environment such as Service Oriented Computing (SOC), grid and cloud.

# Chapter II

# Related Work

Contents



## 1 Introduction

Since we focus on automated negotiation for an efficient cloud provisioning, the related work can be divided into two research areas: (i) Automated negotiation and (ii) Cloud provisioning. In fact, in most cases the two topics have been studied separately. So, in this chapter, we present a review on both cloud provisioning approaches and automated negotiation approaches. In Section 2, we provide a review on cloud provisioning approaches (at business side) and we

pay particular attention on SLA-based provisioning. In Section 3, we propose a review on automated negotiation in utility computing environment: SOC, grid computing and mainly cloud computing. Afterwards, we discuss the limitations of existing negotiation approaches with respect to the research challenges presented in the Introduction. We focus on needs on applying negotiation for an efficient cloud provisioning. Finally, we conclude the chapter in Section 4.

# 2 Cloud provisioning approaches

As mentioned in Chapter I, the cloud provisioning includes three steps: (i) application provisioning, (ii) VM provisioning and (iii) resource provisioning. In this thesis, we focus on application and VM provisioning which are controlled by the business provider. In this section, we present the cloud provisioning approaches including mapping strategy, scheduling algorithms and VM provisioning policies. The mapping strategy indicates the resources needed to execute the required application. The scheduling algorithms determine when and where to execute the request. The VM provisioning policies indicate when a new VM is provisioned. Generally and in most cases, the scheduling algorithm includes the mapping strategy and the VM provisioning policies [20, 26].
We can classify the provisioning approaches into two categories: (1) non-QoS-aware provisioning approaches and (2) QoS-aware provisioning approaches.

## 2.1 non-QoS-aware provisioning approaches

Using non-QoS-aware provisioning approaches, all incoming requests are processed with the same way regardless of their requirements. The scheduling algorithms focus only on resource utilization and do not consider request performance (e.g., response time, completion time). The SLA parameters including price, agreed on QoS and penalties are not considered in the scheduling decisions. In [64], Genaud et al. propose four sets of provisioning strategies where the rented resources are Billed per Time Unit (BTU):

- 1VM4All: One VM is provisioned and all the users' requests are accepted and placed in its queue in order to be assigned to that VM. Using this strategy, the business provider can benefit from an optimal resource cost. But, the wait time will be high which impacts the performance of the requests.
- 1VMperJob-based strategies: At each user's request a new VM is instantiated. There are three versions of *1VMperJob* strategy by reusing idle VMs: (i) *1VMperJobPlus* strategy assigns the request to the first idle VM. (ii) *1VMperJobBest* assigns the request to the

VM having the latest shutdown date. (iii) *1VMperJobBest* assigns the request to the VM having the soonest shutdown date.

- Bin-Packing-based strategies: Those strategies try to optimize idle time by assigning requests to idle resources satisfying a certain condition. If such a VM does not exist then a new VM is deployed. There are three versions: (i) *FirstFit* assigns the request to the first VM where the execution of the request does not require to prolongate the BTU (i.e., no extra cost must be paid). (ii) *BestFit* is an extension of the *FirstFit* where the selected VM leaves the shortest idle time. (iii) *WorstFit* is an extension of the *FirstFit* where the selected VM leaves the longest idle time.

The non-QoS-aware provisioning approaches are not suitable for SLA-based provisioning because if the agreed on QoS parameters are not respected then a penalty must be paid. So, The penalty costs will be high which may impact the provider's profit and reputation. Since QoS parameters are not considered, the above mentioned approaches do not propose a mapping strategy, and the request can be executed in any resource regardless of its requirements.

## 2.2 QoS-aware provisioning approaches

In contrast to the above mentioned works, QoS-aware provisioning approaches have more chance to meet SLA. There are essentially two categories according to the type of resources used for application request execution. The first category considers the same VM type and assumes that all requests have the same execution time. The second category considers heterogeneous VMs, so the request execution time depends on the VM executing it. A detailed description of the approaches in each category is given in the following.

### 2.2.1 Same type of VMs

As extension of Bin-Packing strategies, Genaud et al. [64] propose four scheduling algorithms that consider the completion time: (i) *EarliestFit* assigns the request to the VM where the wait time is minimal. (ii) *RelaxFirstFitx* assigns the request to the first VM while satisfying the following condition "the waiting time do not exceed $x$ times the request runtime". (iii) *RelaxEarliestFitx* and *RelaxLatestFitx* assign the request to the VM that will be idle respectively first and last, while satisfying the condition of the *RelaxFirstFitx* algorithm. Those algorithms consider wait time as factor influencing the completion time. In fact the completion time is a sum of wait time and runtime.

In [24], Leitner et al. propose a scheduling algorithm that tries to find a tradeOff between the application performance offered and the infrastructure cost paid. The objective is to minimize

not only the cost of running VMs but also to minimize SLA violations. The authors propose a greedy approach that calculates the cost of each possible schedule. The cost is calculated as the sum of the infrastructure cost and the penalty cost. The algorithm takes as input the estimated execution time of the request and the current load. That algorithm decides for each request whether to launch a new VM or to schedule it on an existing VM so as the cost is minimal.

In contrast to the above mentioned works which consider that the VM provisioning decision is invoked for each request (per-customer basis), there are some provisioning approaches that adapt the number of deployed VMs according to the demand [7, 65]. Those works focus on VM provisioning rather than requests scheduling. In [7], Calheiros et al. propose an adaptive mechanism for VM provisioning according to the system requirement in terms of response time and number of rejected requests. In fact, the application admission control mechanism rejects requests if all application instances (deployed VMs) have k requests in their queue. The adaptive mechanism is based on information from the *workload analyzer* which estimates future demands. Those information are used by the *load predictor* and the *performance modeler* which solve an analytical performance model. That model is able to predict the effect of a provisioning schedule on system requirement in terms of target QoS (response time, rejection rate). If the estimated values are below agreed on QoS and the resource utilization is under a fixed threshold, the the number of VMs instances is dynamically updated. In [65], Le et al. propose an adaptive resource management approach especially for deadline-bound applications. The objective is to adapt the number of VMs in order to guarantee that request completion time is before the deadline. The authors do not propose an admission control mechanism, all the requests are assigned to a *waiting queue*. Whenever there is an idle VM, a job is dequeued and assigned to that VM. The dynamic provisioning is based on the current status of the *waiting queue* and the VM pool. In fact, the estimated capacity of VM pool can be measured using the *queue theory*. Based on the estimated number of required VMs, new VMs are added or destroyed.

### 2.2.2 Different VMs types

There are several provisioning approaches that consider heterogeneous resources. Those approaches focus on which VMs to execute the request so as to optimize profit and/or customer satisfaction. In fact, the choice of VMs that will execute the requests impacts highly the performance of the request and the resource cost, hence impacts business provider's profit and customer satisfaction. After choosing the appropriate VMs via an efficient mapping strategy, the requests are assigned to those VMs provisioned.

Chen et al. [1] propose a service provisioning approach based on utility-model which characterizes the relationship between business provider's profit and customer satisfaction. The incoming

requests are buffered in a queue. After a certain time interval, the provider runs scheduling algorithm to bid for VM instances able to run the requests while satisfying the provider objective. The authors proposes two scheduling algorithms depending on business provider's objective: (i) optimize profit with a predefined minimal customer satisfaction, (ii) maximize customer satisfaction with a predefined minimal unit profit. The scheduling algorithms include a mapping mechanism which indicates the VM instances that allow to guarantee the customer requirements and respect the resource pricing. The process of scheduling and biding for VM instances is repeated at each time interval. If the request is not completed in the current time interval, it will stay in the queue in order to be processed in the next time interval. In [23], Liu et al. focus on divisible service request and propose a cost-aware scheduling aiming to minimize leased resources cost without SLA violation. After dividing the request into independent and homogeneous subtasks, the proposed algorithm finds the optimal combination of VMs able to run user request subtasks without any SLA violation. The authors use a genetic algorithm, which is not adapted to dynamic environment because the execution time may be very high [25, 26].

Wu et al. [20] propose three types of scheduling algorithms to maximize provider profit while satisfying customer satisfaction through minimizing SLA violations. These scheduling algorithms are designed to minimize the number of instantiated VMs, by maximizing the utilization of VMs that are already initialized. The core of the scheduling algorithms consists of two phases: (i) the *admission control phase*, which evaluates the request and decides whether to accept or reject an incoming request based on the SLA requirements and the resource capability. This phase uses different strategies that guide the evaluation of the request and gives the final acceptance decision; and (ii) the *scheduling phase*, which chooses the scheduling alternative that yields the maximum profit, while scheduling the request according to information that was saved during the admission control phase. The authors in [20] propose three types of scheduling algorithms, based on varying the strategies used in the admission control phase.

In [25, 26], Wu et al. propose a provisioning approach for enterprise software application aiming to optimize the SaaS provider profit and improve Customer Satisfaction Level (CSL) by minimizing SLA violation. In [25], the authors propose a mapping and scheduling policies which minimize cost of rented VMs by maximizing the utilization of instantiated ones. The scheduling algorithms had been extended in [26] in order consider the consumer's future interest in having more accounts by using rescheduling for upgrade service edition request.

There are few works in the literature dealing with negotiation for an efficient provisioning [40, 66]. In [40], the negotiation is carried out between the scheduling service (acting on behalf of the end-users) and the resource manager services (acting on behalf of the providers) in order to find an efficient workflow schedule. The negotiation protocol used is CDA (Continuous double auction) where offers and asks are submitted simultaneously until there is a match or the auction is canceled. The outcome of the negotiation is a set of resources that is provisioned for application

execution. In the same sense, in [66], Prodan et al. propose an advance reservation based on a negotiation approach for application scheduling. The resource manager strategy may be: (1) *attentive*, which consists in generating a new start time the closest to the requested one; or (2) *progressive*, which consists in generating an offer based on other users' requests in order to divide resource equally among users.

## 2.3 A synthesis of cloud provisioning approaches

Table II.1 provides a synthesis of the current cloud provisioning approaches based on the following criteria.

- Cloud provisioning steps (as mentioned earlier we focus on the steps controlled by the business provider): to indicate each steps are addressed: (i) mapping, (ii) scheduling, (iii) VM provisioning.
- Objective: to indicate the business provider objective.
- Profit-aware: to indicate if the business provider provisioning approach consider profit optimization. Since the profit depends on three parameters, that criteria may be split into three sub-criteria on which depends the profit: (i) Budget given by the user for processing the request, (ii) Cost of rented resources where the application requests are executed, (iii) Penalty cost
- Admission control: to indicate if the provisioning approach use an admission control module before accepting a request. The admission control verifies two conditions the feasibility and the profitability.
- Scaling up: this criteria indicates if the provisioning approach allows the instantiation of new VMs (when needed).
- VMs type: to indicate if the same VM type is used (homogeneous), or different types of VMs (heterogeneous).
- Negotiation: to indicate if the provisioning approach uses negotiation or not. If it is the case we indicate in which level the negotiation is carried out: (i) *level 1* between end-users and business providers or (ii) *level 2* between business providers and resource providers.
- Application type: to indicate which type of service is offered (Compute application or Enterprise-based application)

TABLE II.1: A synthesis of cloud provisioning approaches (Business provider side)

| | Provisioning steps:<br>1.Mapping<br>2.Scheduling<br>3.VM provisioning | Objective | Profit-aware<br>a. Budget<br>b. Cost<br>c. Penalty Cost | Admission control | Scaling up | VMs type | Negotiation | Application type |
|---|---|---|---|---|---|---|---|---|
| 1VM4All [64] | 2 | minimize cost | - | No | No | Identical | No | compute |
| 1VMperJob [64] | 2,3 | minimize wait time | - | No | Yes | Identical | No | compute |
| Bin-Packing [64] | 2,3 | minimal cost<br>and maximize rsc utilization | b | No | Yes<br>on-demand | Identical | No | compute |
| EarliestFit [64]<br>Relax-based [64] | 2,3 | minimize cost<br>and maximize rsc utilization<br>and minimize wait time | b | No | Yes<br>on-demand | Identical | No | compute |
| [24] | 2,3 | minimize cost<br>and minimize SLA violation cost | b,c | Feasibility | Yes<br>on-demand | Identical | No | compute |
| [7] | 3 | guarantee the response time<br>and rejection rate | - | Feasibility | Yes | Identical | No | compute |
| [65] | 3 | respect the deadline | - | No | Yes | Identical | No | compute<br>deadline-bound |
| [1] | 1,2,3 | maximize profit<br>or customer satisfaction | a,b | No | Yes<br>spot | heterogeneous | No | compute<br>preemptible |
| [23] | 1,2,3 | minimize cost<br>(without violation) | a,b | no | yes<br>on-demand | heterogeneous | No | compute<br>divisible |
| [20] | 1,2,3 | maximize both profit<br>and customer satisfaction | a,b,c | Feasibility<br>Profitability | yes<br>on-demand | heterogeneous | No | compute |
| [25],<br>[26] | 1,2,3 | maximize both profit<br>and customer satisfaction | b,c | No | Yes | heterogeneous | No | enterprise |
| [40] | 2,3 | Fast and cheap<br>execution of application | b | - | yes | heterogeneous | level 2 | compute<br>workflow |
| [66] | 2,3 | Improve the workflow<br>execution predictability | - | - | yes | heterogeneous | level 2 | compute<br>workflow |

The analysis of the table II.1 highlights the following points:

- Concerning profit optimization, the majority of the elaborated approaches consider only leased resources cost [1, 20, 24–26, 64]. Some of them consider penalty cost [20, 24–26]. Few approaches consider the budget given by end-user for processing the request. In order to optimize the profit, all the parameters on which it depends (budget, resource cost, penalty cost) should be considered as in [20].
- The possibility of instantiating new VMs (scaling up) has been considered by most approaches. Some of the proposed approaches consider identical VMs types [7, 24, 64, 65], some others consider heterogeneous ones [1, 20, 23, 25, 26]. It is important to consider different VMs types since it is more realistic. Furthermore, the execution of a request may vary from one VM to another depending on the performance of that VM, which could impact customer satisfaction. In order to benefit from the attractive resource offers due the competitiveness between cloud providers, it is crucial to consider heterogeneous cloud providers offering different types of VMs as in [20].
- The admission control is used by few approaches [7, 20, 24] despite its effectiveness to guarantee the SLA and to avoid resource overloading. Current SaaS providers do not have admission control and use take-it-or-leave-it strategy. In order to guarantee the SLAs established, some provisioning approaches accept all requests then do adaptation to avoid SLA violation but those approaches are suitable only for same type of requests, and not for heterogeneous requests that need per-customer evaluation and adaptation decision [65]. The feasibility condition is different from one work to another. In [24], the condition checks the possibility of schedule the request by deploying a new VM. However, in [7], the feasibility is checked only on already instantiated VMs. The profitability condition, as part of the admission control, is only addressed in [20]. In order to optimize the profit while guaranteeing the SLA, the two conditions should be verified.
- The majority of the elaborated approaches consider customer satisfaction as a fixed parameter in provisioning. In other words, given the initial user's request, the business provider takes the decision either to accept (even with possible violation) or reject the request. In the same sens, current cloud providers use a take-it-or-leave-it strategy. Those approaches deals with inflexible SLAs. Furthermore, most of those approaches assume that maximizing customer satisfaction is done only via minimizing SLA violation [20, 25, 26]. In [1], the proposed approach is based on variable customer satisfaction, the business provider can either maximize its profit or maximize customer satisfaction. We are interested in an efficient provisioning which leads to maximize both profit and customer satisfaction. From a business provider, those two objectives are dependent and conflicting. Negotiation is considered as the most appealing and flexible approach in dealing with conflicts [22].

While negotiation is considered as in important activity in service/resource provisioning process, the majority of cloud application provisioning approaches do not use negotiation. It is almost used for resource provisioning which be detailed in the next Section. There are few works in the literature dealing with negotiation for improving application provisioning [40, 66]. Those works focus on *level 2* negotiation (i.e negotiation with resource provider) and deals with one-issue negotiation. In [40], the resource prices are generated randomly during the auction. The authors in [66] considers the start time as the only parameter that affects client satisfaction.
In addition to the fact that the SLA cannot be negotiated, the proposed approaches do not consider what to do after an SLA violation, and once established, the SLA cannot be modified.

We propose to handle the last point dealing with inflexible SLA (which is not well addressed yet) by adding negotiation and renegotiation between end-users and business provider during the provisioning process. The other points (profit parameters, heterogeneous cloud providers and admission control) are already addressed in [20]. For that reason, that work has been chosen as basis for our negotiation-based provisioning.

However, to be efficient, the negotiation approaches should be appropriate for cloud provisioning. In what follows, we review the negotiation approaches proposed in utility computing and especially in cloud computing. This is followed by discussing the limitations of those approaches with regards to cloud provisioning properties.

# 3 Negotiation approaches in utility computing

In utility computing, the negotiation is mainly used to establish SLA between consumer and provider with conflicting objectives [67, 68]. In the literature, the negotiation had been addressed in several ways. Some works deals with generic negotiation approaches regardless of the negotiation environment. Some others propose an approach for a given environment (SOC, GRID, CLOUD). In what follows, we give a brief description of generic negotiation approaches and those elaborated in SOC and GRID environment. Afterwards, we give a detailed description of negotiation approaches in cloud environment. Then, we present a synthesis and discussion of the described approaches.

## 3.1 Generic negotiation approaches

As an important step in the SLA life-cycle, the negotiation capabilities are integrated in most SLA management frameworks and languages [9]. The WSLA [69, 70] and the WS-Agreement [71, 72]

are the most used standards and allows negotiation over SLA templates. In [2], the authors give a more detailed comparison between SLA languages considering two negotiation criteria. The first one is related to *meta-negotiation* which expresses the information related to the negotiation establishment. The second one is related to *Negotiability* which represents negotiable SLA terms and their allowed values. According to [2], only WS-Agreement deals with the two criteria and it is the most suitable for negotiation.

The Multi Agent Systems (MAS) are generally used for negotiation [73]. In what follows, we present some relevant generic negotiation approaches within MAS context that can be reused in utility computing environment.
In [74], Tamma et al. focus on a generic negotiation protocol by defining an ontology of negotiation protocols. The ontology is based on common concepts of negotiation protocols and it is shared among participants. In the same direction, in [75], Bartolini et al. propose a general protocol that can be specialized for wide variety of negotiation protocols. The general protocol is based on a taxonomy of negotiation rules (e.g rule for admission of agents, rule for posting a proposal, rules of agreement formation). The authors propose also a framework where participants could negotiate following the general protocol rules. The framework is based on software agent responsible for evaluating negotiation rules and taking actions as a consequence.

In [17], Lai et al. propose a generic multi-issue negotiation strategy aiming to reach a "win-win" solution. The proposed strategy is based on utility function and can be divided into three components: (i) the *conceding* strategy where the negotiator establishes the reserved utility using a time-dependent strategy. (ii) the *responding* strategy responsible for offers evaluation and accept an offer only if the received utility is greater than the utility of the offer that will be sent at the next round. (iii) the *proposing* strategy responsible of counter-offers generation during the negotiation. That strategy is based on the fact that different offers (combination of issues) may have the same utilities. So, it is important to find which offers satisfy both parties. There are two mechanisms for generating counter-offers when the negotiator does not know the opponent's utility function. The first one called *shortest-distance* is used when the negotiator have a utility function. It consists in choosing the closest offer to received offer on the indifference curve/surface. Based on that selected offer, the negotiators can choose a limited number of offers from the indifference curve/surface (i.e those offers have the same utility as the closest offer). The second mechanism called *pareto-optimal mediating* is used when the negotiator does not have an elicited utility function. The mechanism assumes that the negotiator can assess the utility of a limited set of points. Based on those points, the problem can be decomposed into a series of linear negotiation baselines managed by a mediator, which assists the two agents to find a pareto-optimal point.

In [51], Radu proposes an adaptive rule-based negotiation strategy. The rule to be applied at each negotiation round is defined according to previous interactions with other agents in the

system. Given the dynamicity of cloud environment and the high number of users, the approach in [51] is not suitable because of the unlimited number of rules that can be generated.

GENIUS[1] (Generic Environment for Negotiation with Intelligent multi-purpose Usage Simulation) is a negotiation environment implementing an open architecture for heterogeneous agents. It includes a set of agents with a specific negotiation strategies and allows the design and implementation of new strategies by specifying the negotiators' preferences. The strategies are based on negotitors' internal preferences and opponent behaviour. Furthermore, GENIUS allows the simulation of tournaments between negotiating agents. The negotiation follows the bilateral alternating offers protocol [76]. Also, GENIUS includes an analytical toolbox that enables the analysis of negotiation results and the calculation of optimal solutions.

## 3.2 Negotiation approaches in SOC environment

In SOC context, some approaches focus on negotiation frameworks and mainly the decision-making strategies (e.g., [18, 50, 77]) while some others focus on generic (re)negotiation protocol (e.g., [31, 78]).

### 3.2.1 Negotiation frameworks

In [18], Faratin et al. propose a formal negotiation model for service-oriented provisioning. The model focus on decision making process during negotiation. For offer evaluation, the negotiator calculates first the utility value of the received offer's (at point of time $t$). Also, he calculates the utility of the offer that he will send at $t + 1$. Based on those two values, the provider decide whether to accept/reject the offer or else generate a counter offer. The offer generation is a linear combination of simple negotiation decision function called tactic. A *tactic* determines the amount of concession for a single issue at each step by considering a single parameter, such as time, remaining resource, or the opponent's behavior. The authors define three families of negotiation function. The *time-dependent* tactics considers the time given for negotiation and models the fact that negotiators should concede more quickly as the deadline approaches. There are two families of time-dependent functions: polynomial function and exponential ones. The polynomial function concedes faster at the beginning than the exponential one. For each function, the negotiator can generate an infinite number of tactics by varying the function convexity degree parameter, which controls the concession characteristics. That parameter indicates the agent behavior: *boulware* agents make small concession at the beginning (large concession when deadline approaches), *conceder* agents go quickly to their reservation value and *linear* agents

[1] http://ii.tudelft.nl/genius/

concedes evenly during the whole negotiation. A *resource-dependent* tactics considers the resources remaining during negotiation. A negotiator following a *behavior-dependent* tactics will imitate the opponent's behavior and will generate a counter-offer accordingly.

Unlike [18], in [50, 77] the negotiation is done via a broker. In [77], Comuzzi et al. propose a broker representing either provider, or both provider and consumer able to carry out negotiation in semi-automated or automated manner respectively. The broker takes as input the provider/consumer strategy and utility function. The proposed broker uses a concession time dependent function for decision making. In the same sense, in [50], Zulkernine et al. propose an adaptive and intelligent broker able to carry out negotiation between service providers and consumers according to their business level specification. The decision-making strategy is based on parametric time dependent function. The preference parameter is varied according to the received opponent's offer.

In [33, 79], the authors focus on adding negotiation capabilities to the Service Oriented Architecture (SOA). In [79], Hasselmeyer et al. propose an SLA negotiation framework where providers and consumers could negotiate via a negotiation service. The provider is represented by a negotiation service and an SLA template repository comprising the offered services with their possible QoS. The consumers are represented by a negotiation service and a user interface (for human interaction). The negotiation follows the *Discrete-Offer-Protocol* which is a one-round protocol. After the discovery and the selection phase, the consumer sends his request to the concerned provider's negotiation service. The provider checks if there is an offer in the SLA template repository that matches the consumer's request, then decides which offer to send. Finally, the consumer decides whether to accept or reject the request and informs the provider's negotiation service. That work deals with one-round negotiation where all possible offers are stored in advance in the repository, which may be not suitable for dynamic environment like cloud.

In [33], Mach et al. propose a negotiation and re-negotiation framework for web service contracting. The proposed framework architecture is based on the SOA enhanced with negotiation between consumers and providers, where the negotiation and the renegotiation happens before and after service execution, respectively. Each negotiator agent contains a negotiation and re-negotiation engine. That engine is based on a knowledge base that includes three elements: (i) the business rules repository where the negotiator stores the rules implementing his proper business strategy, (ii) the economic cost model where the negotiator stores the cost of each production factor (e.g., computational power, disk space, network bandwidth), (iii) the history data store past information (e.g of successful contracts, statistical data about the QoS of each provider). The re-negotiation happens after SLA violation detected (after service execution).

In [80], Di Nitto et al. propose the architecture of a negotiation framework that can supports various negotiation protocol. The architecture is composed of (i) participants each participant

is represented by a multi-agent system containing one or more negotiators able to carry each negotiation with a manual or automatic manner and a coordinator to manage the negotiations done by each participant. (ii) marketplace which manages the interactions among all participants according to the negotiation protocol. Also, the marketplace contains a mediator able to generate offers on behalf of negotiators. For a bilateral negotiation case and based on the negotiators' evaluations of the generated SLA, the mediator generates proposals so as to fall into the area of the intersection of the two acceptance regions.

In [45], Yan et al. propose an agent-based framework for SLA negotiation aiming at fulfilling end-to-end QoS requirements of the service composition. The authors propose to extend FIPA protocol in order to support one-to-many negotiation (for the same type of service). A Coordination mechanism is proposed to check the overall QoS from the different type of services negotiated. For decision making, two heuristics (concession, and tradeOff) based on fuzzy similarity are proposed. In [81], the agent-based framework is extended to support renegotiation. After establishing all SLAs of the service composition, the QoS values of atomic services are monitored. When an SLA violation is detected, one or more SLAs are renegotiated autonomously in order to guarantee the original end-to-end QoS requirements.

### 3.2.2 Generic negotiation and renegotiation protocol

In [78], Hudert et al. propose a negotiation framework based on WS-agreement supporting a variety of bilateral and multilateral protocols. To do so, the authors define a meta-language enabling the definition of a multitude of protocols. The process of contract creation contains three phase. During the fist phase, an exchange protocol is proposed where the created protocol definition is distributed among prospective negotiators via a third-party coordinator. Afterwards, the participants start negotiation following the rules mentioned at the previous phase. Finally, the negotiation is concluded either by an agreement or not.

Before renegotiation was introduced to the WS-Agreement protocol by the Grid Resource Allocation Agreement Protocol (GRAAP) group [71], many researchers tried to extend the negotiation component of the WS-Agreement standard in order to support renegotiation [30, 31]. Those authors focus on a renegotiation protocol and propose an approach for extending the WS-Agreement standard in order to support renegotiation.

## 3.3 Negotiation approaches in GRID environment

In Grid environment, the majority of negotiation approaches [16, 28, 82] focus on coordinating access to multiple resources from different resource providers. In [82], Czajkowski et al. propose

an SLA negotiation protocol aiming to coordinate access to multiple resource simultaneously. The authors define three types of SLAs: (i) *Task Service Level Agreements* (TSLAs) deals with task performance, (ii) *Resource Service Level Agreements* (RSLA) deals with the right to use resource characterized by its abstract capabilities, (iii) *Binding Service Level Agreements*(BSLA) is for the binding between the reservation of resources (RSLA) and the task (TSLA). That work does not consider resource pricing.

In [28], Ouelhadj et al. propose an SLA negotiation protocol for an agent-based grid scheduling system. The SLA-based grid scheduling infrastructure is composed of three types of agents: the user agent, the local scheduler assigned to a specific computer resource, and the Super scheduler which manager the local schedulers of the same institution. The proposed protocol defines the interactions between those agents and is based on Contract Net protocol (CNP). There are two levels of interactions (i) meta-SLA negotiation between the user and the Super Scheduler (one-to-many) the SLA contain the high level information about the resources needed. (ii) sub-SLA negotiation between the Super scheduler and the local scheduler (one-to-many) the SLA contain the low level information (CPU, memory, etc.). The protocol allows the possibility of renegotiation in case of SLA violation. The proposed protocol is powerful for resource scheduling and infrastructure management but deals with one-round negotiation.

In [3], Venugopal et al. propose a bilateral negotiation mechanism between a resource broker (i.e consumer) and a provider for the advance reservation of compute nodes. The negotiation mechanism is based on Rubinstein's Alternating Offers Protocol. The broker's negotiation strategy is based on estimating the number of nodes required for the application given the application requested deadline. The strategy consists on decreasing the number of required nodes until reaching the lower bound (i.e., the application terminates with deadline) or the provider accepts the request or else the broker receives an offer able to execute the application within the deadline. The provider strategy consists on checking the feasibility of the request respecting the number of nodes and time-slot. If is possible to execute the application, the request is accepted. If it is not possible to respect the number of nodes, the request is rejected. If it is not possible to respect the time-slot, the provider generates an offer containing a new time-slot according to the nodes' availability.

In [83], Sim et al. propose a concurrent negotiation mechanism for grid resource co-allocation ensuring that the consumer can obtain all required resources simultaneously. The negotiation mechanism include essentially two parts: (i) concurrent one-to-many negotiation for one type of resources, where it is possible to renege from a contract by paying penalty fees. This type of negotiation is done by a commitment manager (CE) using time dependent concession making strategy for offer generation. The CE selects the resource contract leading to the highest expected utility, which is based on the reneging probability and penalty fees. (ii) coordinating multiple one-to-many negotiations using a Utility Oriented Coordination (UOC) strategy. The UOC

strategy aims to guarantee a high success rate and a higher utility value. The UOC is based on predicting the change in utility in the next round for each one-to-many negotiation.

In [29], Sharaf et al. propose an extension to the WS-agreement protocol in order to support renegotiation. The authors propose a decision-making strategy based on a fuzzy logic decision support system, as part of the AssessGrid project. The proposed strategy enables the evaluation of an offer received during renegotiation in order to compare it to the original agreement. The authors focus on offer evaluation and do not provide details about offer generation during the renegotiation.

## 3.4 Negotiation approaches in Cloud environment

As mentioned in Chapter I, the cloud provisioning contains two interaction levels. The first level (*level 1*) deals about resource negotiation while the second one (*level 2*) deals about SaaS/PaaS negotiation. Here we will classify existing work according to the provisioning level for which it applies (*level 1* or *level 2*). There are also some works dealing with generic negotiation approaches regardless of the interaction level.

### 3.4.1 Generic negotiation approaches

In [62], Venticinque et al. propose a design of a cloud market called "Cloud Agency" for delivering services and for resource management. The cloud Agency aims to fulfill the requirements of user's applications given a collection of cloud service providers. The architecture is based on software agents having different roles in the market: each provider and each client is represented by an agent. Provider agents are managed by a *mediator agent* which is responsible of selecting the best providers matching the user requirements. The *negotiator agent* is responsible of SLA fulfillment between the mediator and the *client agent*. The authors address the interoperability problem between different cloud providers and users by proposing a unique cloud ontology. In [41, 84], the authors propose the design and development of software agents in a Cloud market with the purpose of enhancing discovery, negotiation and composition of cloud services. The authors adopts the brokering architecture where many brokers act as mediators between consumer agents and resource provider agents. The brokers' role is to sub-lease bundled service to consumers, where the service is composed by multiple resources from different providers. The Cloud market is composed of two interrelated markets (i) service market: where consumers and brokers negotiate using a market driven strategy (time, market factors), (ii) resource market where each broker negotiate multiple providers for reserving different types of resources. The broker consists of a coordinator that coordinates the parallel negotiations done by the commitment managers for acquiring n different types of Cloud resources. Each commitment manager negotiates with

different Cloud providers for one type of resource. That work deals with one issue negotiation (price) and specific scenario where the negotiated service is a group of resources. Furthermore, the negotiation in each market is treated separately from the other one although the dependence between the two markets.

In [32], Hani et al. define three components essential to carry out the renegotiation: (i) the service monitoring able to detect possible failure; (ii) the Service assessment, which defines the renegotiable parameters and the limit for each issue; and (iii) the SLA renegotiation component, which is based on Genetic algorithm. In [85], Galati et al. explore the possibility of adding an SLA renegotiation protocol to CMAC (Condition Monitoring A Cloud) platform. The authors considers the renegotiation as negotiating a counter-offer (before the SLA establishment).

In [86], Silaghi et al. propose a generic time-based negotiation strategy using learning procedure. The proposed strategy aims to maximize negotiators' utilities. Before starting negotiation, the negotiator divide the negotiation time on subintervals. At each negotiation step, the negotiator analyzes the incoming opponent's proposal and updates the opponent profile. Then, for bid generation, the negotiator strategy depends on the subinterval, for instance when the subinterval is close to the negotiation deadline the agents make more concession.

In [22], the authors propose two negotiation algorithms implementing the tradeoff and the concession strategy. The two strategies had been compared in terms of individual utility, social utility and success rate, using a "storage as a service" scenario. The algorithms are valid only for two-attribute negotiation (e.g., storage price, reliability). The amount of concession/tradeoff is identical at each round, and it is fixed before starting negotiation.

Given the potential dependence between service and resource provisioning, the are some works that address the negotiation at the two levels. In [16], Siebenhaar et al. focus on concurrent negotiation in the Cloud market across multiple levels (consumer, service provider, and resource provider). The authors propose a many-to-many protocol based on FIPA specification messages. The many-to-many protocol consists of many one-to-many protocols coordinated by a coordination entity (CE) for each participant in the negotiation. The CE is responsible for the generation of the negotiation entities (NEs) that will handle the one-to-many negotiation. The protocol is composed of two phases. In the first phase (warm-up), the NEs are exchanging pre-proposals and counter pre-proposals until there is at least one acceptable offer. Then, the NE sends a pre-accept to the owner of the best offer and pre-reject to the others. In the second phase (countdown), the NEs are competing to propose the best definitive offer. For decision making the NEs use a time-dependent concession strategy.

### 3.4.2 Cloud resource negotiation (*level 1*)

At this level, the negotiation is about VMs characteristics. The main objective is an efficient resource management while satisfying consumers' requirements. In [21], Son et al. focus on spot instances VMs types. The authors propose a negotiation strategy based on time slot and price preferences in order to maximize resource utilization and minimize resource fragmentation on the provider side and to minimize cost on the consumer side. The authors define a time slot utility function for the consumer and the provider, by converting high-level preferences for each time slot into mathematical parameters. For the consumer, the utility function models preferences for different time slots. For the provider, the utility function is based on prioritization of the set of the available time slot. For each consumer's request, the highest priority is given to time slots with an expected less demand, the earliest time one, and those fitting to the duration demanded. Furthermore, the authors proposes a tradeOff burst mode negotiation strategy where agents are allowed to concurrently make multiple proposals with the same aggregate utility function. This model works only if both consumer and providers can do this conversion, a task which requires skilled users; second, it assumes that the provider can anticipate the demands, which is not simple in a Cloud environment. In [87, 88], Son et al. extend the SLA negotiation approach proposed in [21] with an SLA management approach for an efficient resource allocation in distributed data-centers. In fact, the distribution of virtual machines among servers highly influences the VM performance (response time). So, it is important to find an efficient VM placement in order to maintain the negotiated SLA and to avoid violation. The authors propose and SLA- and location-aware resource allocation scheme. The VM placement strategy considers both the geographical location of data centers and their workload.

In [19], Dastjerdi et al. propose a service negotiation framework facilitating automated SLA negotiation between providers and consumers. The authors propose a concession strategy using a parametric time-dependent function for both consumer and provider. The consumer objective is to concede on less important QoS and verify offers reliability. In fact, the concession parameter for offer generation is defined according to preferences over attributes. the consumer's offer evaluation strategy is based on the reliability of the provider's offer, measured via a third party monitoring service. The provider objective is to maximize profit and balance resource utilization. To do so, the concession parameter is defined according to both resource utilization and preferences over attributes. The concession parameter can be applied only for price generation.

The ANEKA[2] platform presented in [89] supports negotiation for the execution of parallel task application on many nodes. The proposed negotiation strategy for the provider (ANEKA) consists of varying the requested start time according to the availability of nodes.

[2]http://www.manjrasoft.com/aneka$_a$*rchitecture.html*

In [90], Akhani et al. extend Haizea[3] with an advance reservation policy supporting negotiation. The negotiation is launched when the requests of users cannot be fulfilled (i.e., when no more available resource with the needed requirements). The authors propose two algorithms: (i) on the provider side for counter offer generation. It consists in varying at each step some of the user requirements (start time, duration, memory and CPU) in order to minimize resource fragmentation; and (ii) on the consumer side, it consists in selecting the most convenient offer based on requirements' flexibility. The main drawback of that work is that the negotiation strategy is static for all incoming requests and considers only resource fragmentation.

In [52], Copil et al. address the problem of energy consumption in the Cloud. The authors propose a negotiation mechanism aiming to find a balance between energy consumed (i.e amount of resource offered) and performance offered. To do so, the authors propose a negotiation protocol based on particle swarm optimization heuristics leading to a high social welfare. The negotiation process starts by an initialization phase, in which each participant generate a number of possible offers which is called swarm (i.e., the offers closest to their goal). At each round, when an offer is received from the opponent, the swarm evolution is calculated (i.e., the effect of the offer at the current population). Then, the counter offer is generated which is the average of offers in the swarm.

In [91], Stantchev et al. focus on mapping QoS requirements of business process (SLA) to IT infrastructure. The negotiation is carried out between a business process owner and the infrastructure owner. The proposed approach include three steps (i) Formalization where both the requirements of the business process and the capabilities of the infrastructure are formalized. (ii) Negotiation where the business process requirements is compared to infrastructure capabilities under different load hypothesis. The decision where to replicate some service is taken based on that comparison. (iii) Enforcement of business process SLAs at the IT infrastructure level by parallelization of service processing via replication in different datacenters. The proposed negotiation approach is different from classical negotiation ones (where offers and counter offer are generated). In fact, it deals with one-round negotiation with an objective to find an efficient mapping between the business process requirement and resource specifications.

In [46], Marcias et al. propose a negotiation model for cloud resource provider aiming to maximize a non-additive utility function that considers different objectives. The authors define the following goals for the provider: (i) Maximization of revenue (ii) Client classification (iii) Prioritization of tasks in peak-off hours (iv) Maximization of provider's reputation. Each goal can be evaluated using a sub-utility function based on SLA terms requested. The evaluation parameters are collected from resource level information (historical monitoring data, resource status) Given

[3]Open source VM lease manager: http://haizea.cs.uchicago.edu/

the importance of each goal (weight), the provider utility is calculated as a weighted sum of goals' sub-utilities. The price and time slot are generated so as to have the maximum utility.

In [92], An et al. focus on the resource allocation problem in a dynamic cloud market. The authors extend the Rubinstein's alternating offers protocol in order to include tentative agreement where either buyer (i.e consumer) or seller (i.e provider) can cancel the agreement without paying penalty. The authors propose two negotiation strategies: (i) on the buyer side (i.e., consumer side) where the proposed price for the resource depends on pressure deadline (for acquiring the resource), the seller's expected cost of providing the resource, the demand/supply ratio of resource over time (ii) on the seller side (i.e., provider side), by trying to maximize its revenue by accepting offers, decommitting from agreements, or even by canceling some tentative agreements.

### 3.4.3 SaaS/PaaS negotiation (*level 2*)

The negotiation at this level is between end-users that want to use applications already deployed on the Cloud and business provider. The negotiation is about application performance characteristics.

In [14], Wu et al. propose a negotiation framework based on a broker which assists consumers to find SaaS providers satisfying their needs. The broker's objective is to optimize its profit margin and to satisfy the user's requirements. The provider's aim is to maximize its revenue by accepting the maximum number of users. The strategy used for counter offer generation is time-dependent and market-dependent.

In contrast to [14], in [15], the negotiation is carried out directly between agents representing consumers and providers. In that work, Son et al. propose a mechanism that adaptively controls negotiation issues weights by analyzing workload trends. During a negotiation session, the preferences (negotiation issues' weights) are changed based on workload prediction. The idea is to propose attractive prices and to postpone requests in peak load period in order to avoid SLA violations. The authors define a utility function for each issue (price, response time, time slot). The overall utility is a weighted sum of the individual utilities. For counter offer generation, the agents use a time-dependent concession strategy.

In the context of the SLA@SOI project, in [10], Yakub et al. propose a robust and computationally inexpensive negotiation strategy aiming to reach a near-optimal SLA. The negotiation is performed by a negotiation manager component that have been integrated to the business layer. The negotiation manager is able to negotiate with both SaaS users and IaaS providers (negotiation at level 1 and level 2), During negotiation, the agents follows the bilateral alternating offers

protocol. For decision-making, the agents use a concession strategy called *Reactive Exploitation*, where the degree of concession is defined according to the opponent behavior.

## 3.5 Synthesis of negotiation approaches

Table II.2 and II.3 provide a synthesis of the current negotiation approaches in SOC/Grid and Cloud environment respectively. The synthesis is based on the following criteria.

- Objective: to indicate the objective of the related work dealing with negotiation.
- Service type: to indicate the type of the service negotiated.
- Objects: to indicate the negotiation issues.
- Protocol: to indicate the protocol followed during negotiation.
- Technique: to indicate the technique used by the negotiation decision-making strategies.
- Decision factors: to indicate the elements considered when evaluating and generating offers during negotiation.
- REnegotiation: to indicate if there is a possibility of renegotiate an already established contract. If yes, we specify if the renegotiation is reactive or proactive.
- Implementation: to indicate if the proposed approach is implemented or not.

In addition to the above mentioned criteria, in table II.2, we specify the context (SOC or Grid). Furthermore, for the negotiation approaches in cloud environment (Table II.3), two additional criteria are considered:

- Multi-layer: to indicate if the dependence of service provisioning on resource provisioning is considered.
- Heterogeneity: (i) Protocol: to indicate if the heterogeneity of protocols is considered, (ii) Service description: to indicate if the heterogeneity of service description is considered.

TABLE II.2: A synthesis of negotiation approaches in SOC and Grid environment

| | Objective-Contribution | Service type | Objects | Protocol | Technique | Decision factors | Renegotiation | Implementation | Context (SOC,Grid) |
|---|---|---|---|---|---|---|---|---|---|
| [18] | Propose a formal negotiation model for service-oriented provisioning | Service | - | (bilateral) Alternating offer | Concession | Preferences Time Remaining resource Opponent's behavior | no | yes | SOC |
| [77] | Extend SOA to support negotiation | Service | - | (bilateral) Alternating offer | Concession | Preferences Time | no | no | SOC |
| [50] | Propose an adaptive and intelligent negotiation broker framework | Web service | - | Bilateral defined protocol | Concession | Preferences Time Opponent's offer | no | yes | SOC |
| [79] | Extend SOA to support negotiation | Service | - | Discrete-Offer | Static match | - | no | yes | SOC |
| [33] | Extend SOA to support negotiation | Web Service | - | Alternating offer (one round) OR auction | - | Business rule Cost model History data | reactive | no | SOC |
| [80] | Propose an SLA negotiation framework | Service | - | - | Simulated Annealing | - | no | yes | SOC |
| [45] | Fulfill end-to-end QoS requirements of the service composition | Composite service | - | (many one-to-many) FIPA Iterated Contract Net | Concession tradeOff Fuzzy similarity | Preferences Opponent's offer | no | yes | SOC |
| [81] | Fulfill end-to-end QoS requirements of the service composition | Composite service | - | (many one-to-many) FIPA Iterated Contract Net | Concession tradeOff | - | proactive | no | SOC |
| [78] | Support a multitude of protocols based on WS-agreement | Service | - | defined protocol | - | - | no | no | SOC |
| [30], [31] | Add renegotiation capabilities to WS-agreement | Service | - | defined protocol | - | - | yes | no | SOC/Grid |
| [82] | Propose a negotiation protocol aiming to coordinate access to multiple resource | Grid resource | - | defined protocol | - | - | no | no | Grid |
| [28] | Propose an SLA negotiation protocol for an agent-based grid scheduling system | Grid resource | - | (one-to-many) Contract Net protocol | - | - | reactive | no | Grid |
| [3] | Propose a negotiation mechanism for the advance reservation of compute nodes | Grid resource | Start time | (bilateral) Alternating offers | - | Available nodes | no | yes | Grid |
| [83] | Propose a concurrent negotiation mechanism for grid resource co-allocation | Grid resource | Price | (one-to-many) Defined protocol | Concession | Reneging probability Penalty fee Time | no | yes | Grid |
| [83] | Extend WS-Agreement with renegotiation | Grid resource | - | - | - | Historical data | yes | yes | Grid |

TABLE II.3: A synthesis of negotiation approaches in Cloud environment

| | Objective-Contribution | Service type | Objects | Protocol | Technique | Decision factors | RE negotiation | Multi layer | Heterogeneity | | Impl. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | Protocol | Service description | |
| [62] | Design of a cloud market | Cloud service | - | - | - | - | yes | no | yes | yes | no |
| [41], [84] | Design and development of a cloud market | Cloud service | Price | multilateral | Concession | Preferences<br>Time<br>Market factors | no | no | no | no | yes |
| [16] | Propose a concurrent negotiation approach across multiple levels | Cloud service | - | multilateral | Concession | Preferences<br>Time | no | yes | no | no | yes |
| [32] | Propose a renegotiation framework | Cloud service | - | - | Genetic algo | - | proactive | no | no | no | yes |
| [86] | Propose a generic time-based negotiation strategy using learning procedure | Cloud service | - | bilateral | Concession<br>Learning | Preferences<br>Time<br>Opponent profile | no | no | no | no | yes |
| [22] | Propose two negotiation algorithms implementing the tradeoff and the concession strategy | Cloud service | - | bilateral<br>Alternating offers | Concession<br>Tradeoff | Preferences | no | no | no | no | yes |
| [21],[87] [88] | Propose a price and time slot negotiation strategy | IaaS | Price<br>Time slot | bilateral<br>Alternating offers | TradeOff (burst mode) | Preferences | no | no | no | no | yes |
| [19] | Propose an SLA negotiation framework | IaaS | Price | Extended alternating offers | Concession | Preferences<br>Rsc utilization<br>Reliability | no | no | no | yes | yes |
| [89] | Propose a negotiation mechanism for the execution of parallel task application on many nodes | IaaS | Start time | - | - | Nodes availability | no | no | yes | yes | yes |
| [90] | Extend Haizea with an advance reservation policy supporting negotiation | IaaS | Number of nodes,<br>Memory,<br>CPU<br>Start time,<br>Duration | - | - | Resource status | no | no | no | no | yes |
| [52] | Propose a negotiation mechanism aiming to find a balance between energy consumed and performance offered | IaaS | Price,<br>Memory, Cpu | bilateral<br>Alternating offers | Particle Swarm Optimization (PSO) | Preferences<br>Time<br>Opponent Offer | no | no | no | no | yes |
| [91] | Mapping QoS requirements of business process requirements to IT infrastructure | IaaS | - | - | Comparison | Infra capabilities,<br>Service replication | no | no | no | no | yes |
| [46] | Propose a non-additive utility function that considers different objectives | IaaS | Price<br>Time slot | Alternating offers | - | Historical monitoring data<br>Resource status | no | no | no | no | yes |
| [92] | Design a negotiation model for the problem of dynamic resource allocation in a cloud market. | IaaS | Price<br>Penalty | Extended<br><br>alternating offers | Consumer:<br>Concession<br>Provider:<br>Optimization | Consumer:<br>Time<br>Resource cost<br>Demand/supply<br>Provider:<br>Resource status<br>Resource cost<br>Penalty | no | no | no | no | yes |
| [14] | Propose an automated SaaS negotiation framework | SaaS | Price<br>Refresh time<br>Process time<br>Availability | one-to-many<br>Alternating offers | Concession | Preferences<br>Time<br>Market factors | no | no | no | no | yes |
| [15] | Propose an adaptive negotiation strategy according to workload change | SaaS/PaaS | Price<br>Response time,<br>Time slot | bilateral<br>Alternating offers | Concession | Preferences<br>Time<br>Workload status | no | yes | no | no | yes |
| [10] | Propose a negotiation strategy based on opponent behavior | SaaS/PaaS | Availability<br>Performance<br>Backup | bilateral<br>Alternating offers | Concession | Preferences<br>Time<br>Opponent behavior | no | yes | no | no | yes |

## 3.6 Discussion

In recent years, SLA negotiation has received great importance. Despite active research, several limitations can be identified according to the research challenges presented in the Introduction. The first challenge deals with cloud provisioning in general. The other challenges are related to SaaS provisioning which had been chosen as an application domain.

- How the negotiation should be designed for an efficient cloud provisioning

  The Cloud provisioning has two main properties that highly impact the efficiency of negotiation. The first one is the dependence of cloud application provisioning on resource provisioning from the IaaS layer (multi-layer). The second property is the dynamic context of provisioning. In fact, there may be many elements that impacts the quality and the price of the service to be provisioned. Those elements are denoted by decision factors. Most of literature focuses on negotiation in specific layer and primarily resource negotiation at the IaaS layer. A negotiation approach dealing with specific layer cannot be suitable for cloud provisioning where there is two interdependent interactions levels. The multi-layer property is insufficiently taken into consideration. The elaborated negotiation approaches in SOC and grid context do not consider the multi-layer property because the nature of those environment do not require the multi-layer aspect. Furthermore, there are few works [10, 15, 16] in the cloud dealing with the multi-layer aspect (see Table II.2). In [16], the authors focus on a multi-layer negotiation protocol without considering the negotiation decision-making strategies. In [10, 15], the negotiation is specific to SaaS/PaaS negotiation and cannot be applied across levels.

  Most of the elaborated negotiation approaches deals with a decision-making strategy that uses a fixed technique considering fixed elements (i.e., decision factors). The most used technique is concession and the most used decision factors are the time and the negotiators' preferences. Those strategies are generally efficient just for a given negotiation scenario. For instance, the strategies based on learning technique [86] are not suitable for an ever-changing environment. Furthermore, AI techniques that aims to search optimal or even sub-optimal solutions [52, 76, 80] are not suitable for dynamic environment. In fact, formalizing all the elements impacting the quality of the service and its price, and the relation between them may lead to intractable optimization. Even generic decision-making strategies [76] are relevant for specific situation and cannot be applied for dynamic scenario. In fact, those strategies are based on the representation of the large space of possible contracts. Furthermore, the negotiator's preference should be fixed before starting negotiation [17].

  Given the cloud dynamicity, when the situation changes, the specific strategy may not be relevant. So, we need a generic negotiation strategy that considers the dynamic elements

related to the provisioning. In addition to negotiators' preferences, market factors and opponent offers; there are elements related to the provisioning process such as allocation and monitoring. Those elements impacts highly the quality of the service to be provisioned and its price. Furthermore, the monitoring information can serve for renegotiation which is very important in such dynamic environment. Despite their importance, those elements are insufficiently taken into consideration. In fact, while most of IaaS negotiation approaches considers resource allocation, only few service negotiation approaches [15] considers resource allocation. Furthermore, the elements related to the monitoring are considered only by [19, 46].

There are many negotiation approaches relying on broker architecture [62, 93] mainly in SOC context [50, 77, 80]. Due to cloud characteristics (dynamicity and highly scalability), we need decentralized solution to handle negotiation.

To the best of our knowledge, according to the state-of-the-art no generic multi-layer (re)negotiation model taking into account dynamic elements had been proposed in SOC, Grid and Cloud environment.

- How SaaS application negotiation could maximize customer satisfaction and optimize business provider profit

  As mentioned in the previous Section, the majority of SaaS provisioning approaches do not consider negotiation between end-users and business provider (see Table II.1). Furthermore, unlike IaaS and cloud service negotiation which have been deeply studied, SaaS/-PaaS negotiation are insufficiently developed in the literature [10, 14, 15]. The IaaS negotiation approaches as well as the Grid resource negotiation approaches cannot be applied to SaaS/PaaS negotiation, because the strategies proposed are related to resources characteristics which are different from application one. In addition, Cloud service negotiation approaches as well as service negotiation approaches in SOC context present the negotiation as an independent activity from the others activities of the provisioning process. To be efficient, the negotiation must consider others provisioning activities such as resource allocation, which is responsible for service execution. In fact, the allocation including scheduling and VM provisioning impacts highly the quality of the service to be provisioned. Also, the provider profit depends on the resources leased and their prices. So, in order to maximize customer satisfaction to optimize the profit, the business provider should consider when negotiating both requests scheduling and VM provisioning.

  The elaborated SaaS/PaaS negotiation approaches are not designed for customer satisfaction maximization and profit optimization. In [14], the resources' costs are not considered when calculating the total provider's revenue. Also, the information related to request placement in virtual resources (i.e., scheduling information) are not considered by the negotiation decision-making strategies. Those information impacts highly the offered performance which impacts customer satisfaction. In contrast to [14], in [15], the business

provider's strategy considers resources' costs and workload status but the proposed strategy is not suitable for users having high urgent requests. In fact, the strategy is based on postponing users' requests in peak load period. Also, the proposed approach assumes a fixed resource pool with the same users' requirements.

Although the fact that negotiators' preferences can be dynamic, in [10], the authors assume that the preferences are fixed before starting the negotiation and are the same for all incoming requests. Furthermore, the offered price is not considered when generating possibles offers. It is important to consider the price, because the performance offered depends on the price paid which depends also on the market status.

To the best of our knowledge, according to the state-of-the-art, there is no SaaS negotiation approach that optimizes business provider's profit and maximizes customer satisfaction.

- How the business provider could maximize the number of clients given the constraints related to resource availability and pricing at negotiation time ?

  The majority of the negotiation approaches focus on bilateral negotiation without considering the concurrency among clients' requests. In fact, the bilateral protocol is used in most approaches and the concurrency is absent ad decision factor (see Table II.2, Table II.3). There are some negotiation approaches dealing with multilateral negotiation (e.g., [16, 19, 41]), those works focus on choosing best provider(s). However, in order to maximize the number of clients, the business provider should be able to assign efficiently available resources among concurrent requests.

  In addition, the variation in resource availability and pricing are insufficiently taken into consideration by some proposed decision-making strategies (e.g., [21], [15]. In fact, those approaches consider the variation just during negotiation session which is limited by the deadline. In those mono-session negotiation approaches, if the negotiation fails, the users' requests are rejected. Due to the gap between users' requirements and the provider capabilities, this may lead to an increased number of rejected users.

  To the best of our knowledge, according to the state-of-the-art, there is no adaptive and concurrent SaaS negotiation approach that considers variation in resource availability and pricing.

- How to overcome the problem of unexpected events leading to potential SLA violation

  Most of the SLA-based application provisioning approaches discussed in the previous Section assumes that once an SLA is established, it cannot be renegotiated [23–26]. Unlike SLA negotiation, the concept of SLA renegotiation has not yet been well studied [27].

  There is some work that tries to enhance the WS-Agreement negotiation protocol in order to support renegotiation [29–31]. In [30, 31], the elaborated approaches do not consider decision-making strategies during renegotiation. In [29], only the offer evaluation strategy

is considered, and there is no detail about how the offers are generated during renegotiation. Also, the proposed protocol in [28] allows renegotiation in case of SLA violation, but the proposed approach focus on negotiators' interactions before SLA establishment.
In [81], the elaborated renegotiation approach is specific to service composition scenario.
In [32, 33], the authors propose general conceptual renegotiation frameworks without giving details about the renegotiation decision-making strategies. In [62], the authors present some issues related to renegotiation without presenting a concrete contribution on how it could be done.
To the best of our knowledge, according to the state-of-the-art, there is no decision-making model that guides the renegotiation process toward a satisfactory agreement.

# 4 Conclusion

In this chapter, we have analyzed and discussed related work on cloud provisioning and negotiation. The analysis of cloud provisioning approaches shows that SLA negotiation between end-users and business providers is insufficiently taken into consideration. In fact, most of cloud provisioning approaches deals with inflexible SLAs. Furthermore, the analysis shows that although the service/resource negotiation problem has been widely treated in literature, there are still some limitations that have to be considered (e.g., multi-level, dynamicity).

In this thesis, we aim to overcome these limitations. To deal with multi-level and and dynamic environment, we propose a generic multi-layer (re)negotiation model taking into account dynamic elements that can influence the negotiation outcome. Furthermore, in order to optimize business provider's profit and maximizes customer satisfaction, we propose a negotiation-based provisioning approach. Also, to deal with the variation in resource availability and pricing, we propose an adaptive and concurrent SaaS negotiation approach aiming to maximize the number of clients. Finally, we propose a proactive renegotiation approach aiming to handle potential SLA violation. The different contributions of the proposed approach are discussed in the next chapters.

# Chapter III

# The negotiation-based framework applied to cloud provisioning

Contents



## 1 Introduction

For efficient cloud provisioning, both providers and users should be satisfied in spite of their conflicting needs. In contrast to take-it-or-leave it strategies, negotiation is a flexible solution to solve conflicts and enable reaching satisfactory agreements for both parties.

Cloud provisioning has two main properties that highly impact the decision-making process of negotiation and strategies for reaching a satisfactory agreements for both parties. The first one is the dependence of cloud application provisioning on resource provisioning from the IaaS layer. The second property is the dynamic context of provisioning, where the context refers to all elements in relation with the quality and the price of the service to be provisioned. Among those elements there are some related to the provisioning process itself (e.g., scheduler, resource provisioner, Monitor). Also among others elements we can cite those related to the negotiator's preferences, market situation, etc. The importance of those elements vary according the negotiation situation, and the information given by those elements are constantly varying during a negotiation session.

Despite their importance, the two properties mentioned above are not well addressed in the current negotiation models. Most of literature focuses on negotiation in one specific layer and primarily resource negotiation at the IaaS layer [13]. Furthermore, the relation between the negotiation and the other provisioning activities is not explicitly presented. For resource negotiation, there are some works that consider resource allocation (e.g., [21], [3]). However, the application negotiation is presented as an independent activity from the application provisioning process. The decision-making strategies (even generic ones) are suitable for specific situation, and cannot be applied for various scenarios.

Our objective is to design a generic negotiator suitable for cloud environment, that can be used by any cloud actor (user, business provider, resource provider) and for any negotiation scenario (with possible dynamic changes). To do so, we propose a generic negotiation-based provisioning process which contains the most important activities to be considered for an efficient negotiation-based provisioning in the cloud. Furthermore, the process shows how negotiation is guided by the provisioning activities, especially allocation and monitoring steps. The proposed process shows the different cases where it is important to renegotiate either before SLA establishment or after SLA establishment.In this thesis, we focus on the negotiation and renegotiation activities as part of the overall provisioning process. We propose also design of generic negotiator that can be instantiated in each cloud layer and able to act on behalf any cloud actor (user, business provider, resource provider). The negotiator's decision-making strategy takes into account dynamic elements in relation with the provisioned service QoS and its price. The decision making strategies are designed in a way to consider potential dynamic changes and to acts accordingly. Based on the negotiator model and its instantiation among cloud layers, we propose a multi-layer and dynamic negotiation framework for cloud provisioning.

This chapter is organized as follows. In Section 2, we present our negotiation-based provisioning process. Section 3 details the negotiation and renegotiation activities by presenting the negotiator model and its behavior. Then, we show how the model can be instantiated among cloud layers

resulting in an operational multi-layer negotiation framework presented in Section 4. Finally, Section 5 concludes the chapter.

## 2 Overview of the generic negotiation-based provisioning process

The proposed negotiation-based provisioning process is based on the SLA-based provisioning process illustrated in Figure I.4 (see Chapter I). The SLA-based provisioning process is composed of six activities: Discovery, selection, negotiation, allocation, monitoring and assessment and termination. We focus essentially on the negotiation and the renegotiation and their relation with the others provisioning activities. Our objective is to show how the negotiation is guided by the provisioning activities which impact the quality of the provisioned service. This is done via the design of a generic decision-making model, which considers not only the negotiator internal elements (preferences, related negotiation, etc) but also external elements that can be gathered from the provisioning activities.

As shown in Figure III.1, the negotiation-based provisioning process is composed of seven activities: Discovery, selection, meta-negotiation, negotiation and renegotiation, allocation, monitoring and assessment and termination. As compared to the SLA-based provisioning process, we have added the meta-negotiation activity. Also, we define the relation between the activities for an efficient provisioning. In the following, we describe the modification added to the original process by highlighting their importance for an efficient provisioning.

The Cloud computing is an heterogeneous environment; there is no standard protocol for communication between clouds nor a common semantic for expressing resources and their characteristics. So the negotiation between heterogeneous parties may lead to inconsistencies. For that reason, negotiators should at least agree on the negotiation protocol and SLA terms semantics before starting the negotiation. This activity is called meta-negotiation [89], [2], [94], [95] and aims to facilitate communication between heterogeneous parties. It is done before starting the negotiation process by agreeing on information related to the negotiation establishment. During this phase the negotiators define the information related to the negotiation establishment (e.g the negotiation protocol, the negotiation issues, the authentication method reference) [2]. We believe that the meta-negotiation is a crucial activity in the provisioning process; in fact negotiators may have different protocols and may express their SLA terms differently. Consequently, it is essential to consider a meta-negotiation activity to avoid inconsistencies and to reach a common understanding during negotiation. So, in order to address the heterogeneity of protocols, we assume that there is library of protocols including FIPA protocols (which provides a formal

definition of several negotiation protocols) and possible other commonly used protocols such as alternating offer protocol.

Given the negotiation protocol and the SLA terms, the negotiation between the user and the provider can be started. Due to the gap between user's requirements and the provider capabilities at negotiation time, the negotiation may not result in a compromise satisfying both parties. Since negotiation is already limited by deadline (the generation of offers and counter-offers cannot be infinite), the negotiation may fail. So, in that cases, we propose that the provider may launch a renegotiation, when there are new elements that may change the outcome of the previous failed negotiation. That type of renegotiation is described in Chapter V.
If the negotiation succeeds, the SLA containing the agreed on terms is established. Consequently, based on the SLA the provider allocates the adequate resources (virtual or physical) based on the SLA established.

The monitoring activity should be able to report standard performance measurements to the provider, which could anticipate system degradation and so trigger a renegotiation before SLA violation and so avoid paying high penalties. The resource provider can monitor physical resources, and can detect if the VM running on those resources may be impacted. For example, the business provider can monitor the state of VMs where application are executing. By doing so, if there are problems (VM failure, VM performance degradation), the provider can predict potential violation and so trigger renegotiation proactively (if allowed by the other party). That type of renegotiation is described in Chapter VI.

As explained above, the renegotiation may be triggered in two cases: (1) Before SLA establishment when the first negotiation fails, (2) After SLA establishment when the monitor risk of violation. If the renegotiation fails, it can be repeated or not depending on the condition defined by the two parties before starting the negotiation. Intuitively, the renegotiation after SLA establishment cannot be repeated, and the provider must respect the new SLA since a second violation will be critical for his reputation. But, the renegotiation before SLA establishment can be useful as long as the user does not find the required service/resource.

We design a generic decision-making model that can be used in negotiation as well as in renegotiation. For that reason, we refer to negotiation and renegotiation activity by a single term (re)negotiation that will be used throughout this chapter. The decision-making model is based on the decision-making strategy responsible for offer evaluation and generation. The decision-making strategy should be guided not only by internal elements (e.g., negotiator's preferences) but also external elements that can be gathered from the provisioning activities. There are two important activities: (i) Allocation activity which includes the mapping, the scheduling and the resource provisioning. In fact, for the request evaluation and for offer generation, the provider should know the requests' translation in terms of required resources (mapping), also the current

state of resources and the requests assigned to those resources (scheduler). Also, the provider should know if there is possibility to have additional resources (resource provisioning). (ii) Monitoring and assessment activity which can inform the provider about the new system status after dynamic changes. So, the provider can renegotiate according to those information. In the next chapters, we will study in depth how the information given by the allocation activity and the monitoring activity are used by the negotiation approaches for an efficient application provisioning. In this thesis, we focus on the negotiation and the renegotiation activities ((re)negotiation)

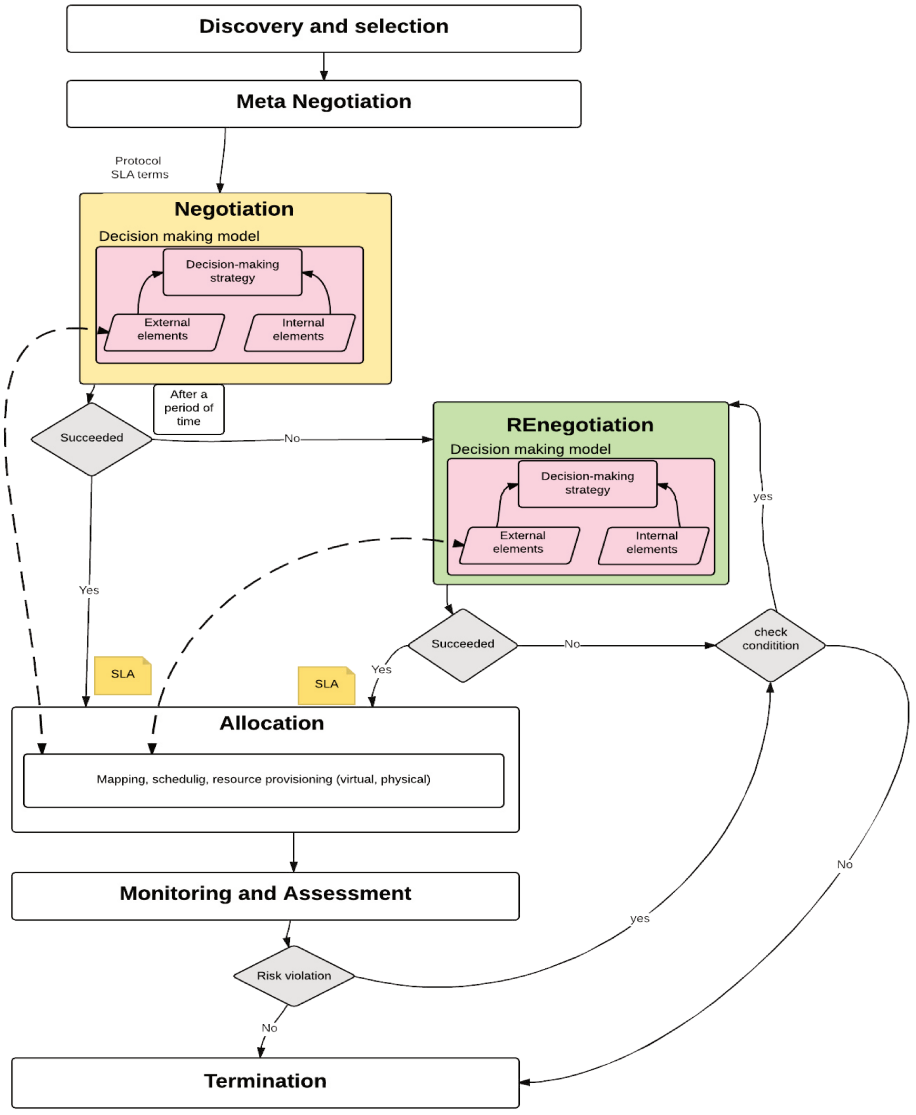


FIGURE III.1: Generic negotiation-based-provisioning process

as part of the overall provisioning process. For that reason, we suppose that the discovery and the selection are done. We assume also that the negotiation protocol and the negotiation issues are know between negotiators before starting negotiation (meta-negotiation). In what follows, we will give a detailed description of the (re)negotiation activity.

# 3 Description of the (re)negotiation activity

The (re)negotiation activity can be described by giving the negotiator model and the behavior process of the negotiator during the (re)negotiation.

## 3.1 The negotiator model

Figure III.2 illustrates the main concepts in relation with a *negotiator* in the cloud. A negotiator is able to carry automated negotiation on behalf of cloud actor (e.g end-user, business provider or IaaS provider). Generally the negotiator is represented by an agent. The negotiator may have one or two roles. A *role* is either a consumer or a provider. A negotiator with two roles is both a consumer and provider, for example, the business provider is an application provider and at the same time a consumer of resource from the resource layer. Each role may follow a *negotiation session* based on a *decision-making strategy*. When a negotiator has two roles, it is important to define a coordination mechanism that manages the communication between the two roles. In what follows, we will detail each concept defining the negotiator.

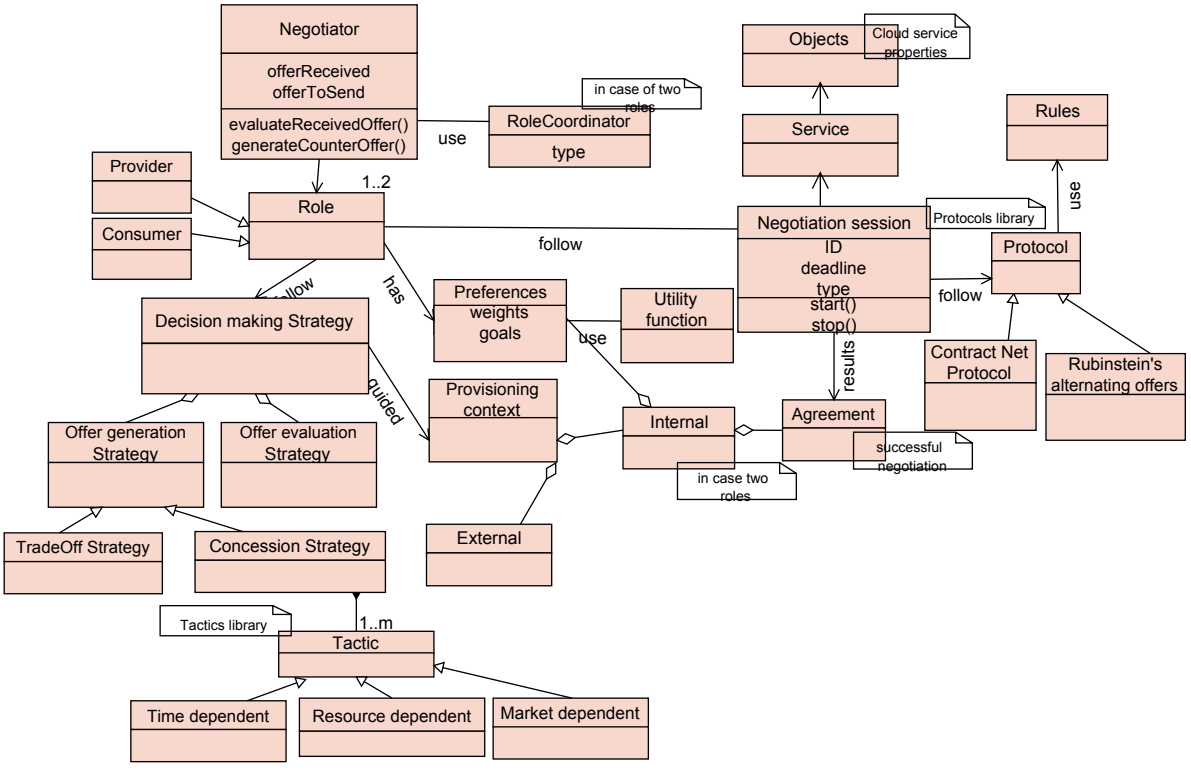


FIGURE III.2: A generic negotiator model during the (re)negotiation phase

### 3.1.1 The negotiation session

A negotiation session can be defined as the period covering the time from when the interaction between negotiators begins until the interaction ends (with agreement or not) in order to get/offer a cloud service. The negotiated *service* (e.g., software, resource) is composed of *objects* (also called issues) over which the actors negotiate. Generally the negotiation issues are related to the service QoS and price. In fact, there is a relationship between the service performance and the its price. Negotiable SLA terms in the Cloud differ from one layer to the another, except for generic issues such as price. For each type of service, there correspond some characteristics that are the object of negotiation. A negotiation session follows a protocol that defines the interaction rules between participants and defines the condition for termination. Fipa Contract net protocol and Rubinstein's Alternating Offers Protocol are the most common used protocols for negotiation. The details about protocol types are given in the Appendix A. We are interested especially in multi-round protocols where the negotiators could exchange offer and counters-offer at each round.

### 3.1.2 The decision-making strategy of the negotiator

The (re)negotiation activity is based essentially on the negotiator decision-making model. An efficient decision-making strategy is the key of successful (re)negotiation. During a negotiation session, each negotiator follows a decision-making strategy, which is composed of:

**Offer evaluation strategy:** This allows a negotiator to evaluate an offer and to decide whether to accept, reject or propose a counter-offer according to the predefined preferences of the negotiator. The preferences are generally given by the cloud actor. The strategy is generally based on a *utility function* that measures the degree of satisfaction for each offer that is received. Based on the received utility value, the negotiator decides about the action to take (accept/reject/counter-offer). In the case of negotiating with multiple opponents, the negotiator chooses the offer with the highest utility value. The utility-based evaluation strategy is efficient when the cloud actor could formally define the utility function and his preferences. While this could be possible for an end-user, but for cloud providers it would be difficult to formulate all the elements related to the negotiated service. In fact, those elements may impact the action taken at each round. For example, to evaluate an incoming application request, the business provider must check the capabilities in terms of owned resources, also the possibility to get more VMs and others factors that impact the quality of the provisioned application. For that reason, we believe that the offer evaluation should be guided by the provisioning context, which refers to all elements in relation with the quality and the price of the service to be provisioned.

**Offer generation strategy:** This enables a negotiator to generate offers and counter-offers at each round. There are essentially two approaches to generation of counter-offers: *tradeoff* and *concession*. During a negotiation session, a negotiator may choose to use the concession approach, to use the tradeoff approach, or to combine them. The *tradeoff approach* consists of decreasing the utility values of some issues and increasing the values of others, in order to keep the overall utility value unchanged. The *concession approach* consists of decreasing the value of the utility function (i.e., the satisfaction) using one or a combination of tactics. A tactic is represented by a *negotiation decision function* which defines the amount of concession at each negotiation round [18]. The *negotiation decision function* types are explained in Chapter I. Time-dependent, resource-dependent and market-dependent are the most appropriate functions for negotiation in the cloud environment. In fact, the time, the remained resource and the market factors are important elements impacting the QoS of the service and its price. For that reason, we have reused those tactics in our model (stored in a *tactic library*). But, those tactics could be used only if the negotiator knows in advance the reserved offer. Fixing a reserved offer before starting negotiation cannot be possible mainly for cloud providers. This is due to dynamic factors such as resource availability and dynamic pricing. So, in those cases, the offer generation should be based at least on the state of resources and their costs. For that reason, we believe that the offer generation should be guided by the provisioning context.

As explained above, both offer evaluation and offer generation strategies should be guided by the cloud *provisioning context*. We define the provisioning context as the elements that could impact the quality of the provisioned service and its price. We define two types of contexts:

- Internal context: The internal context contains the information private to the negotiator. It is composed essentially of the preferences and the related negotiations. The preferences are used as inputs by a negotiator and the preferences are generally communicated by the cloud actor. Examples of preferences are: the reserved and the preferred value for each issue, the importance of each negotiation issues (weight), the goal of negotiation (maximize profit, maximize number of clients, etc.). The related negotiation contains the output of the negotiation carried by the other role (in the case that a negotiator can have two roles). In the latter case, the output will be used not only for multi-layer negotiation, but also for renegotiation.
- External context: The external context contains the information given by the external modules in relation to the negotiated service. The external modules can be related to the provisioning activities (scheduler, VM provisioner, monitor) as mentioned in the previous section or others modules like the market prospector. The market prospector estimates the number of competitors and alternatives of the concerned service and can also guide the

pricing during negotiation. The offer generation strategy can be seen as a combination of tactics selected from the tactic library and new ones based on the external context. An example of that strategy we will be detailed in the next chapter for SaaS application provisioning scenario.

To summarize, the proposed decision-making model can be used for any negotiation scenario by varying the elements related to the context considered. Also, the information gathered from the provisioning context can assist the negotiator to choose the suitable tactics for offer generation (from the tactic library). For example, when there are many competitors (as communicated by the market prospector), the negotiator should choose a market-dependent tactic. Also when the time for negotiation is important, the negotiator should choose a time-dependent tactic.

### 3.1.3 The coordination between interdependent negotiation sessions

The *role coordinator* is used only in cases where the agent has two roles. It is important to define the coordination between the two interdependent negotiation sessions, each triggered by a given role. We consider three types of coordination for a business provider agent:

- The agent negotiates first with the end-user in the context of its provider role, and then the agent negotiates with the IaaS provider in the context of its consumer role, and based on the output of the prior negotiation with the end user in the context of the provider role. That coordination may be not efficient when the business provider does not succeed to obtain the needed resource (from the IaaS provider) guaranteeing the already established SLA with the end-user.
- The agent negotiates in the context of the consumer role against the IaaS provider (when possible), regardless of the demand. The agent then negotiates in the context of the provider role based on the available resources already negotiated in the earlier negotiation (no synchronization between the two roles). That type of coordination may lead to an over-provisioning of resource compared to incoming load.
- The agent negotiates in both roles simultaneously. In the provider role, the agent begins negotiations with the end user; in the consumer role, the agent negotiates with IaaS provider based on the end-user request; and then the agent communicates the result to the provider role in order to carry out the negotiation with the end user.
  In what follows, for the operational framework, we have chosen the third type of coordination (negotiating in both roles simultaneously) because it allows to acquire just the needed resources.

## 3.2 The behavior process for the negotiator

When receiving an offer during a negotiation session, the negotiator has a generic behavior for decision-making. We model the behavior process using the *Rubinstein's Alternating Offers Protocol*. The protocol followed during the negotiation session imposes the messages exchanged between the negotiators. The proposed generic behavior process of a negotiator $r$ is described by the following expression:

$$BP_r(O_{p->r}, feval_r, fgener_r, SLA).$$

The definition of the variables of the formula is listed in Table III.1. The behavior process $BP_r$ is described in Figure III.3.

TABLE III.1: Description of symbols

| | |
|---|---|
| $p$ | The proposer $p$ |
| $r$ | The responder $r$ |
| $O_{p->r}(t)$ | The offer $O$ sent from negotiator $p$ to negotiator $r$ at time t |
| $SLA(O)$ | The generated SLA contract corresponding to the offer $O$ |
| $feval_r$ | The evaluation function of negotiator $r$ |
| $fgener_r$ | The function that generates one or more offers from negotiator $r$ |
| $t_{deadline}$ | The negotiation deadline |

Figure III.3 shows the behavior process triggered by an offer $O_{p->r}(t)$ received from a proposer $p$. The negotiator starts by evaluating this offer using the evaluation function $feval_r$ as part of the *offer evaluation strategy*. The action taken by the responder depends on the evaluation function output. The action will be one of the following three cases.

- *accept*: If the received offer is acceptable, the two parties establish the SLA contract $SLA(O_{p->r}(t))$.
- *reject*: The responder can reject the offer, and so the negotiation session terminates without an agreement.
- *propose*: The responder can propose one or more counter-offers using the offer generation function $fgener_r$ specified in the *offer generation strategy*. Then the agent $r$ starts a new round $t+1$. If the negotiation deadline $t_{deadline}$ is reached, the negotiation terminates without an agreement. Otherwise, the responder sends the counter-offer $O_{r->p}(t+1)$.

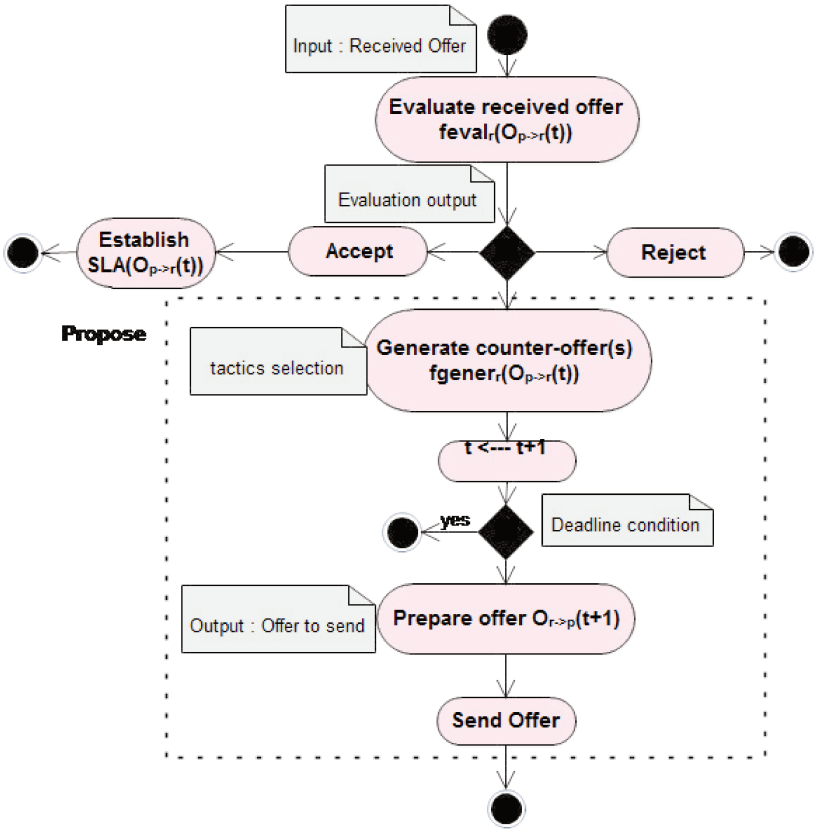


FIGURE III.3: Overview of the negotiator behavior during (re)negotiation

Note that the proposed generic behavior for decision making may be adapted to any negotiation protocol by modifying the messages exchanged during the negotiation session. The overall interaction between one proposer and one responder during a negotiation session is composed of two phases:

1. The initiation phase: A negotiator starts by sending one or more offers. This phase may be triggered by the consumer or by the provider, according to the protocol being followed in the negotiation session. For example, for an auction protocol the provider initiates the negotiation session. This is in contrast to the alternate offer protocol and the contract net protocol, in which the negotiation session is initiated by the consumer.

2. The iterative offer exchange phase: This phase is guided by the protocol rules. The two negotiators exchange offers and counter-offers using the generic behavior process either until one party accepts or rejects the opponent's offer or until the negotiation deadline is reached.

# 4 Multi-layer and dynamic negotiation framework for cloud provisioning

In this section, we present an overview of the negotiation framework for cloud provisioning. The framework is based on the negotiator model described in the previous section. The objective is to enhance the classical provisioning process by adding negotiation capabilities between actors. We adopt the agent paradigm in order to implement autonomous software entities able to represent cloud actors. A detailed description of Multi-Agent Systsem is given in Appendix B. Cloud actors represented by software agents could negotiate in a dynamic and flexible manner. In fact, a software agent can perceive the environment changes and acts accordingly using his own internal state. In our context, the environment changes are examples of cloud dynamism; the agent actions are the possible reaction of negotiators and the internal state is the agent's own preference. Each actor is represented by an agent based on the negotiator model described in Figure III.2. The negotiation agent should be able to carry out negotiations according to the internal and external contexts. As described in Section 3.1. The internal context contains the information which is private to the negotiator (e.g., preference, past negotiations). The external context contains the information from external modules in relation with the QoS and the price of the service to be provisioned.

The proposed multi-layer and dynamic negotiation framework is illustrated in Figure III.4. The agent model (i.e the negotiator) is instantiated for the three cloud layers: user layer (NegoUser), business layer including PaaS and SaaS layers (NegoBusiness) and resource layer (NegoIaaS). In

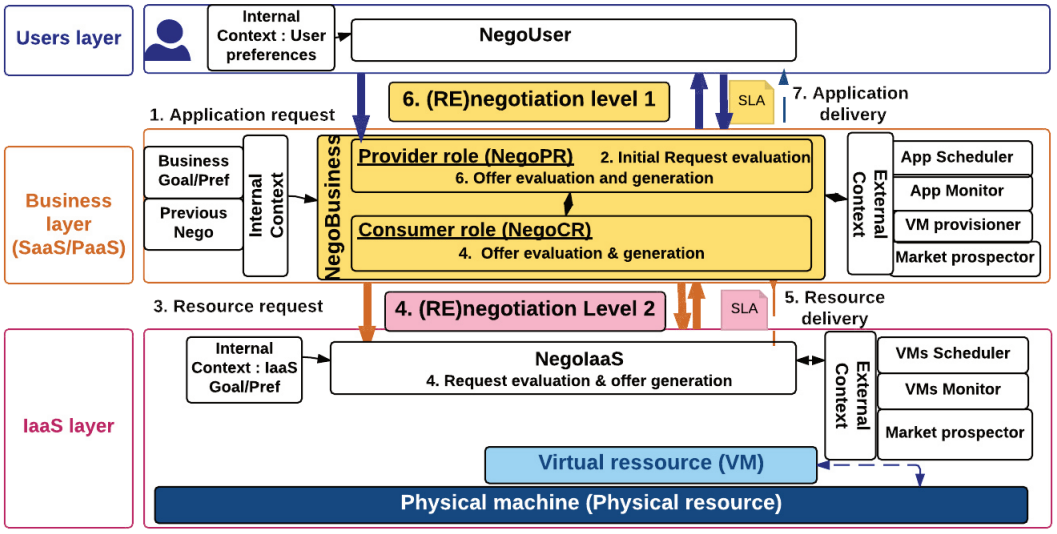


FIGURE III.4: Multi-layer and dynamic negotiation framework for cloud provisioning

order to define our agent-based framework, we detail bellow the agents, the environment, and the possible interactions between agents for an application provisioning:

## 4.1 The agents

We consider three type of negotiation agents:

*NegoUser* **agent:** The *NegoUser* agent has a consumer role and acts on behalf of the end-user and under its control. The end-user gives as input: its preferences about issues (for example, the end-user may give more importance to the price than the response time), the utility function ($feval$) which evaluates offers, and the function ($fgener$) for counter-offer(s) generation.

*NegoBusiness* **agent:** The *NegoBusiness* agent represents the business provider. The provider may have two roles: (i) the provider role, denoted *NegoPR*, which may negotiate with the *NegoUser* for application provisioning; and (ii) the consumer role, denoted *NegoCR*, which may negotiate with the resource provider represented by the *NegoIaaS* agent for VM provisioning. The decision making (offer evaluation and counter-offer strategies) for each role is guided by the internal and external provisioning context. The most relevant elements to consider for the internal context are: the business provider preferences; the goal; and the previous negotiations for VM provisioning. For the external context, as mentioned in the previous Section 3.1, the *NegoBusiness* may use information from the application scheduler, the application monitor, the VM provisioner and the application market prospector. Those modules have a large impact on the quality of the service to be provisioned and its price.

*NegoIaaS* **agent:** This agent acts on behalf of the IaaS provider. It has as internal context the IaaS provider preferences. For the external context, the *negoIaaS* may communicate with the following modules for an efficient decision-making during negotiation: the scheduler of the VMs, the monitor of the VMs, and the resource market prospector.

## 4.2 The environment

The agents are negotiating in a cloud marketplace; each actor belongs to a cloud layer according to the type of service offered. For efficiency in decision-making, the agents communicate with the modules representing their external context (the scheduler, the monitor, or the market prospector). Additional modules that may impact the provisioning may also be added to the external context, in order to be considered in the decision-making.

## 4.3 The interaction among agents during application provisioning

During the negotiation-based provisioning process, the negotiators exchange offers based on the generic behavior described in Figure III.3. The offer is composed of a set of attributes corresponding to the value of the QoS parameters, which may be negotiable or not. The negotiable ones have values that may change over time during a negotiation session, in contrast to the non-negotiable terms, which have fixed values. There are two levels of interaction: 1) *Level 1* deals with the interaction between the users layer and the business layer and 2) *Level 2* contains the interactions between the business layer and the IaaS layer (resource layer). The negotiation-based provisioning process is composed of two phases defined according to the coordination between the two roles of the business provider described in Section 3.1.3.

### 4.3.1 The request phase (from the user layer to the IaaS layer)

The request phase consists of the following steps:

**Step 1:** the $NegoUser$ starts the process by sending a request to the $NegoBusiness$ containing his/her requirements in terms of QoS and price.

**Step 2:** the $NegoPR$ evaluates the incoming request, by mapping it to the needed resource(s) able to execute it while respecting the QoS requirements. For the evaluation, the $NegoPR$ considers the already available resources among the existing ones (i.e already acquired by the *negoBusiness* from the IaaS layer) and can also consider potential additional ones (based on the output of $NegoCR$ negotiation). Those information concerning the resources (already acquired and additional ones) are given by the application scheduler and the VM provisioner. If the request can be scheduled and the QoS requirements can be fullfilled, the evaluation module returns true. And otherwise it returns false. Other conditions may be checked during the evaluation such as the profitability for a given request (i.e if the request acceptance brings some "monetary" profit to the provider).

**Step 3:** If the available resources (among existing ones) do not satisfy the incoming request's QoS requirements, the $NegoCR$ should ask the $NegoIaaS$ for new resources.

**Step 4:** $NegoIaaS$ evaluates the request of the $NegoCR$ according to the available physical resources and according to its IaaS resource allocation strategy. If it is possible to provide the VM(s) with the required characteristics, the evaluation module returns true. Otherwise it returns false.

### 4.3.2 The delivery phase (from the the IaaS layer to the user layer)

The delivery phase depends on the result of the evaluation. In the traditional process for provisioning, if the evaluation returns true, the request is accepted and the delivery succeeds (*level 1* and *level 2*). Otherwise, the request is rejected (according to the take-it-or-leave-it strategy). We enhance the provisioning process with a possible negotiation at each level mainly when the the evaluation returns false. The new delivery phase consists on the following steps.

**Step 5:** The $NegoIaaS$ and $NegoCR$ negotiate for resource delivery, until reaching an agreement satisfying both parties, or until the deadline is reached (*level 2*). If the negotiation succeeds, an SLA is signed between the two parties.

**Step 6:** The IaaS provider delivers the requested resource to the business provider if the negotiation succeeds at Step 5.

**Step 7:** The $NegoPR$ negotiates with the end-user concerning application delivery (*level 1*), and the SLA is signed if the negotiation succeeds.

**Step 8:** If the negotiation succeeds at step 7, the business provider allocates the needed resources and starts the execution of the end-user's request.

## 5 Conclusion

In this chapter, we have proposed a generic negotiation-based provisioning process highlighting the interaction between the negotiation and others provisioning activities. Then, we propose a generic negotiator model able to act on behalf of any cloud actor (end-user, business provider, resource provider) and for any negotiation scenario. That model was instantiated among cloud layers resulting in a multi-layer (re)negotiation framework for cloud provisioning.

For the next contributions, we focus on the first level negotiation (i.e., between end-user and business provider) and more specifically for compute-intensive SaaS provisioning scenario while taking into account varying IaaS providers. In the next chapter, we propose a bilateral negotiation-based provisioning approach for compute-intensive SaaS application.

# Chapter IV

# Bilateral negotiation for an efficient SaaS application provisioning

Contents

# 1 Introduction

With the growth of competitiveness in cloud, business providers must ensure an efficient provisioning which maximizes customer satisfaction and optimizes their profit. For a business provider, an efficient provisioning is not a trivial task when considering different IaaS providers and heterogeneous users' requests (with different requirements). Indeed, for each incoming request, the provider must take the right decision about its placement on rented VMs, while satisfying the QoS requirements and maximizing profit. To optimize the profit, the business provider should maximize the number of clients while minimizing the costs of rented VMs. In order to guarantee that the SLA is met, the business provider must adopt profit- and QoS-aware provisioning approach.

To serve several clients while respecting the established SLAs, some proposed QoS-aware provisioning approaches [7] adapt the resource pool according to the load prediction and not per-customer basis (coarse-grained reasoning). Furthermore, those approaches assume that users have the same requirements. So, they are not suitable for heterogeneous requests mainly with different deadline constraints. In fact, those requests need a per-customer basis provisioning (fine-grained reasoning). As a solution, in [20], an admission control strategy is proposed to decide whether to accept or to reject the request based on the resource capacity and request profitability. When it is not possible to schedule the request, the request is rejected in order to avoid SLA violation. By considering the constraints of resource provisioning from the IaaS layer (cost, availability, etc.), the business provider may reject several requests, which may lead to loss in profit.

Negotiation-based approaches are promising solutions when dealing with conflicts. Although its importance, SaaS application negotiation has not yet been well studied [13]. Current SaaS providers use take-it-or-leave-it strategy. In [14], the proposed approach (as the most approaches elaborated for service negotiation) presents the negotiation as an independent phase from the service provisioning activities. In fact, in the decision-making strategies, they do not consider the SaaS provisioning context including the scheduling and the VM provisioning, although their impact on the QoS of the service to be provisioned and its price as explained in the previous chapter.

In this chapter, we propose a bilateral negotiation approach between business provider and end-user within a compute-intensive SaaS provisioning. The proposed approach aims to maximize customer satisfaction and optimize provider's profit. When no scheduling solution is possible, the business provider can propose an alternative schedule, satisfying the user by harnessing the tradeoff between SLA attributes. By doing so, the provider can win more clients, which leads to an increased profit. We conduct simulation to assess the negotiation-based approach. The

experiments show the benefits of adding negotiation to the provisioning process by improving the business provider profit, the number of accepted users' requests, and the client satisfaction.

This chapter is organized as follows. In section 2 we describe the compute-intensive SaaS provisioning context. In section 3, we show how we integrate negotiation capabilities to the provisioning process. The negotiation decision-making strategies are detailed in section 4. Section 5 presents experiments to assess the bilateral negotiation approach for SaaS provisioning. Finally, in Section 6, we conclude the chapter.

## 2 Compute-intensive SaaS application provisioning scenario

As explained in previous chapter, the application provisioning includes two levels of interaction: (i) *level 1* where an SLA is established between end-user and business provider and (ii) *level 2* where an SLA is established between business provider and IaaS provider. In this section, we present the SLA model at each level and the assumptions made for our scenario. Then, we present the negotiation framework for SaaS provisioning as a specialization of the multi-layer framework illustrated in Figure III.4.

### 2.1 SLA models and assumptions

**SLA model at *level 1*** At this level, the SLA attributes depend on the SaaS application type. For compute-intensive applications, the request may contain the following attributes: Deadline, Budget, Request size, Penalty Rate, file Input size. For this work, we assume that deadline and budget issues are negotiable and other issues are not. In fact, the budget that will paid be by the user and the deadline offered are the key issues for customer satisfaction. Also, for simplification reasons, we will consider only the deadline ($D$), the budget ($B$) and the request size ($RS$) as parameters in the request/offer evaluation and generation. In addition, we assume that the request size is expressed in Millions of Instructions (MI). Thus, the request processing time ($procT$) will depend on the Million Instructions Per Second (MIPS) offered by the VM executing it.

**SLA model at *level 2*** The request may contain the following terms: VM Initiation Time or service provisioning time ($iniT$), Price, Processing Speed, Input Data Transfer Price, Output Data Transfer Price, and Data Transfer Speed. We use MIPS as a measure of the processing power of the VM.
We assume that the VM provisioning model is on-demand, as this is the most common model [23], and the VM pricing model follows that of cost-per-hour billing. The resource provider

charges the SaaS provider based on the pricing model, the VM type and the time used. We assume that the interaction at this level excludes negotiation (all terms are non-negotiable).

## 2.2 Overview of the (bilateral)negotiation framework for SaaS application provisioning

Based on the above cited assumptions and the proposed multi-layer framework (Figure III.4), we propose the (bilateral)negotiation framework for SaaS application provisioning illustrated in Figure IV.1. Here we focus essentially on the Business provider's modules with which the business agent *NegoBusiness* interacts. As explained in the previous chapter, those modules are parts of the external provisioning context. The internal context will be described in section 4. The *NegoBusiness* uses information from the modules responsible for application provisioning.

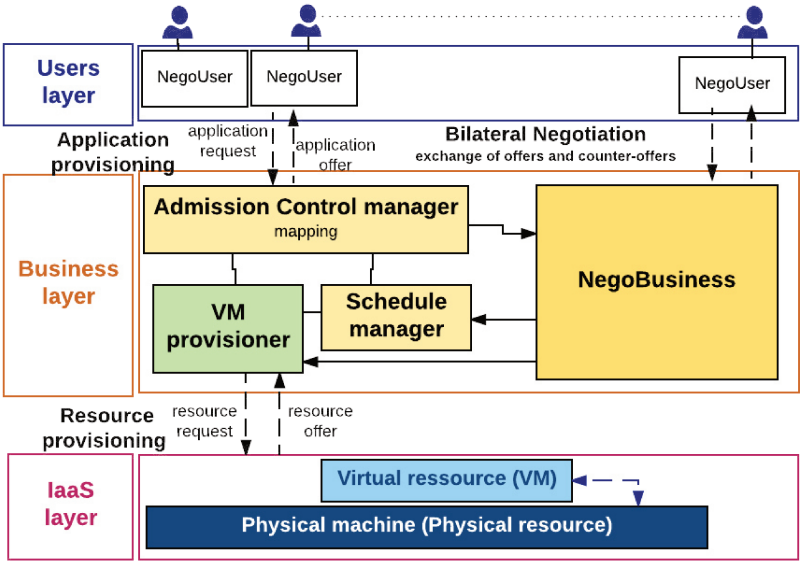


FIGURE IV.1: (Bilateral)negotiation framework for SaaS application provisioning

Here, we have chosen the provisioning approach proposed in [20] because: (i) they are SLA- and profit-aware; (ii) they consider heterogeneous and dynamic resource provisioning from different IaaS providers; and (iii) they perform efficiently when evaluated and compared with reference algorithms. A more detailed explanation and comparison to the current provisioning approach are provided in Chapter II.

Based on that provisioning approach, there are three modules responsible for SaaS provisioning:

- Admission control manager: is responsible for request mapping and evaluation. The evaluation is based on information about current available resources and potential acquired

ones. Those information are used to make the decision to accept or to reject the request. To be accepted, the request must verify two conditions: (i) *The profit condition*: the requested budget must be greater than the overall cost. In fact, to process a request, the SaaS provider may pay resource cost and also potential SLA violation cost. (ii) *The deadline condition*: the request must be completed before the requested deadline. The estimated completion time can be calculated based on the request length and the the VM capacity that will execute the request.

- schedule manager: is responsible for assigning users' requests to available time slots. It has information about the load of each owned resource.
- VM provisioner: is responsible for VM provisioning and deploying application instances on that VM. It has information about resources for lease (BTU price, initiation time, provider owner, capacity, etc.). That information is used when the provider would like to scale up.

The *NegoBusiness* is able to carry out negotiation with end-users when it is not possible to schedule their requests. The offers generated during the negotiation are based on information given by the admission control manager. Since we exclude negotiation with the resource provider, so we focus only the provider role of the *NegoBusiness*. If the negotiation at *level 2* was considered, the consumer role will negotiate with the IaaS providers and then informs the VM provisioner to instantiate the VM and deploy the application.
In what follows, we describe how we add negotiation capabilities to the provisioning approach, by detailing the interaction between the *NegoBusiness* and the above mentioned modules. The negotiation aims not only to increase the business provider's profit, but also to increase the satisfaction of the consumers.

# 3 Negotiation-based SaaS application provisioning approach

In this section, we present first how we extend the admission control strategy in order to support negotiation-based provisioning. Then we present the interaction between the *negoBusiness* and the business provider's provisioning modules.

## 3.1 Towards flexible admission control process

we present first how the request is evaluated using the initial algorithm (on which we base our work). Then we present our extension in order to support negotiation.

### 3.1.1 Classical admission control process

Based on all the studied strategies in [20], Figure IV.2 models their generic common process: the strategy consists first in checking if the deadline is greater than the estimated completion time (*deadline condition*) and then checking if the request's profit is greater than the minimum expected profit (*profit condition*). If one condition is not satisfied, the admission control process returns false and rejects the request. Otherwise, the scheduling information are stored in a list called *potential scheduling list*. The calculation method of the request's estimated completion

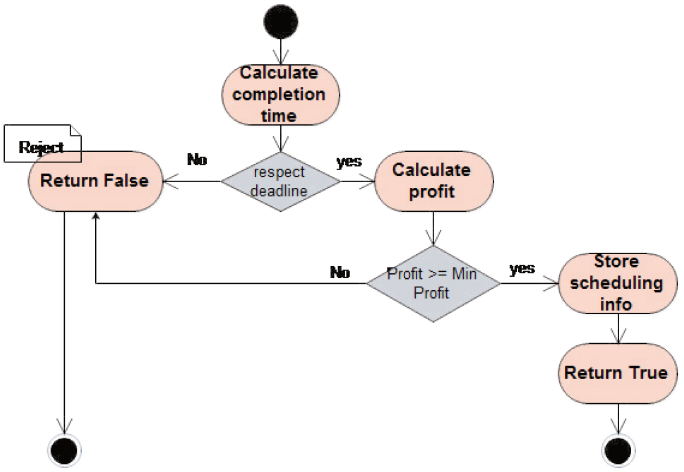


FIGURE IV.2: Admission control strategy process

time (*compTime*) and the request's profit (*profit*) depends on the scheduling strategy. The example below shows an admission control process using two strategies : 1) *'canwait'* strategy checks the possibility of waiting all already accepted requests, 2) *'canInitiateNew'* strategy checks the possibility to create a new VM (scaling up).

**Example of an admission control process** The admission control process first checks if it is possible to schedule the request on an existing VM using the *'canwait'* strategy. If it is not possible to wait on all initiated VMs, then it checks if it is possible to initiate a new VM using the *'canInitiateNew'* strategy.

If the provider schedules the request on an already initialized VM, the user must wait for all previously accepted requests on that VM to be completed. So the estimated wait time is equal to the sum of the processing times of the already accepted requests, minus the execution time incurred by the currently executing request. If the provider launches a new VM, the new request has to wait during the provisioning time of the VM, which is not immediate. This situation is

summarized in (IV.1).

$$compTime_i = \begin{cases} \sum_{k=1}^{n} procT_k - a(j) + procT_i, & \text{if exist VM} \\ initT + procT_i, & \text{if new VM} \end{cases} \tag{IV.1}$$

Where $procT_k$ is the $request_k$ processing time, $a(j)$ is the execution time already spent by the currently executing request , and $iniT$ is the VM initiation time. If the *deadline condition* is satisfied, the provider calculates the profit gained by processing the request using the equation (IV.2).

$$profit_i = budget_i - infCost_i - penaltyCost_i \tag{IV.2}$$

Where $Budget_i$ denotes the received price for serving $request_i$, $infCost_i$ and $penaltyCost_i$ denote respectively the infrastructure cost and penalty cost. The infCost is calculated by equation (IV.3) and it depends on the scheduling strategy decision, if the request will be scheduled on a new VM, we add the initiation cost.

$$infCost_i = \begin{cases} procT_i * price, & \text{if exist VM} \\ iniT * price + procT_i * price, & \text{if new VM} \end{cases} \tag{IV.3}$$

The penalty cost represents how much the user pays when the provider misses the deadline. It depends on the penalty model adopted. Here we assume that the cost will be proportional to the delay, it is calculated by the following formula (IV.4):

$$penaltyCost_i = penaltyRate_i * delay_i \tag{IV.4}$$

With $penaltyRate_i$ denotes the ratio for user compensation when missing the deadline and $delay_i$ is the variation between the agreed on deadline and the actual response time.

### 3.1.2 Flexible admission control process

We aim to enhance the admission control phase in order to minimize the number of rejected requests. The logic of our solution is when it is not possible to schedule the request respecting the initial SLA, the admission control manager stores the reasons of rejection as parameters to use them later in negotiation. For each incoming request, the provider calculates two parameters:

1. The flexible time (fTime): It is calculated using the formula (IV.5).

$$fTime_i = deadline_i - compTime_i \tag{IV.5}$$

Where $deadline_i$ is the initial requested deadline and $compTime_i$ is the completion time and is computed using equation (IV.1). The *fTime* is referred by the *missing time* if the deadline requested is lower than the estimated completion time and which is otherwise referred by the *extra time*.

2. The flexible budget (fBudget): It is computed using the following equation:

$$fBudget_i = profit_i - minExpectedProfit_i \tag{IV.6}$$

Where $profit_i$ is calculated using the equation (IV.2) and $minExpectedProfit_i$ denotes the minimum expected profit and is generally defined by the provider proportional to the incurred cost. The *fBudget* is referred by the *missing budget* if the budget requested is lower than the budget needed to have the minimum expected profit and which is otherwise referred by the *extra budget*.

The flexible admission control process is depicted in Figure IV.3. The process starts by calculating the flexible time (*fTime*) and the flexible budget (*fBudget*). When one of the two conditions is not satisfied (*deadline condition* or *profit condition*), the provider stores the *fTime* and the *fBudget* (i.e., flexible scheduling info) on a list called *Negotiation potential Scheduling List*. If the two conditions are satisfied, the process returns True (i.e., accepted request). Thus, contrary to the classical admission control which saves only possible scheduling alternatives, the *Negotiation potential Scheduling List* saves all failed scheduling alternatives in order to use those alternatives later during the negotiation.

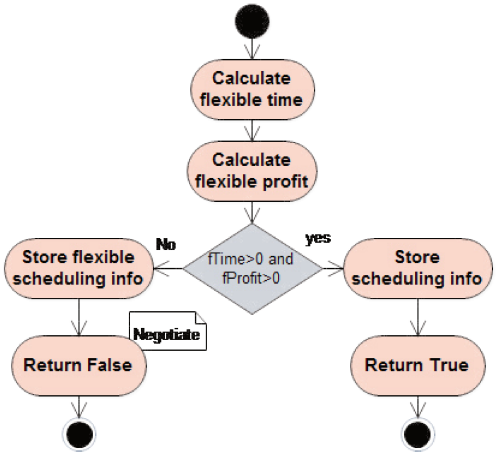


FIGURE IV.3: Flexible admission control strategy process

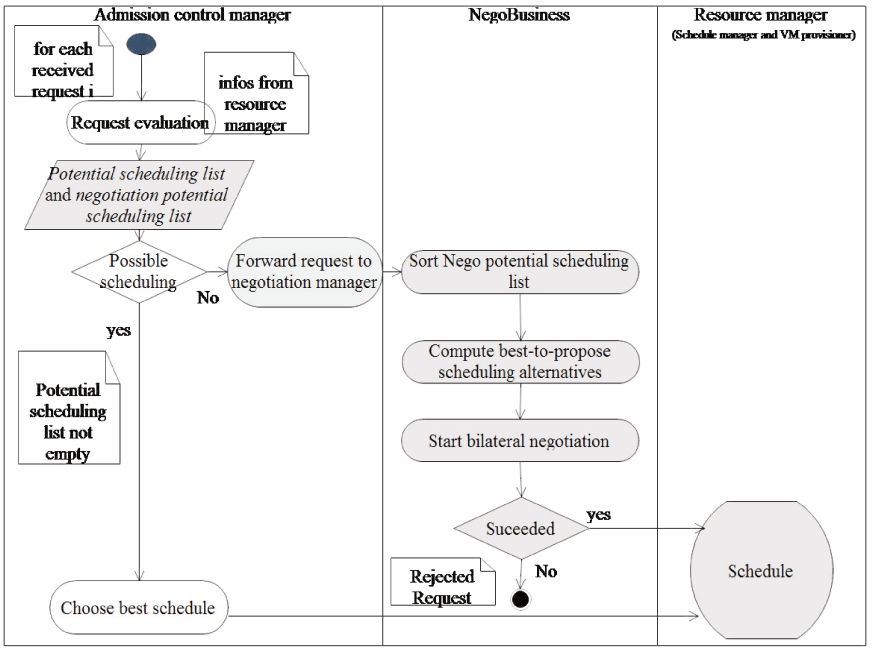


FIGURE IV.4: Interaction between the *negoBusiness* and the business provider's provisioning modules

## 3.2 Interaction between the *negoBusiness* and the business provider's provisioning modules

Figure IV.4 shows the interaction between the negotiation agent *negoBusiness* and the business provider's provisioning modules. As mentioned earlier, the modules are: Admission control manager, schedule manager and VM manager. For simplicity, in the figure, we represent both the schedule manager and the VM provisioner by the resource manager (i.e the resource manager is responsible for scheduling and VM provisioning). The extensions we made are presented in Grey boxes while white boxes present the already existing activities in the classical provisioning process on which we base our work. The process starts by evaluating the user's request via the admission control manager using the flexible admission control strategies. The flexible admission control phase has two outputs (1) the *potential scheduling list* and (2) the *negotiation potential scheduling list*. If it is possible to schedule the request (the *potential scheduling list* is not empty), our process is the same as the original one. The admission control manager chooses the scheduling alternative that yields the maximum profit. If the best schedule include an instantiation of a new VM, this is done via the VM provisioner. Then, the request is scheduled according to the information stored in the *potential scheduling list* via the schedule manager. Contrary to the classical provisioning process which rejects the user's request when the *potential scheduling list* is empty, our process tries to propose to the user others offers that he could schedule via the *negoBusiness* agent.

First, the *negoBusiness* sorts the *negotiation potential scheduling list* by generating three sub-lists based on the parameters ($fTime$, $fBudget$) that were stored during the flexible admission control phase: 1) $subList_1$ contains potential offers with extra time and missing budget; 2) $subList_2$ contains potential offers with missing time and extra budget; and 3) $subList_3$ contains potential offers with missing time and missing budget. We assume that each potential offer $p$ is stored in the sub-list(s) as follows: ($D_{op}$,$B_{op}$,$fTime_p$, $fBudget_p$) where $D_{op}$ and $B_{op}$ denotes respectively the deadline and the budget that can be offered as counter-offer. The user request is denoted by $(D_r, B_r, RS_r)$ where $D_r$, $B_r$ and $RS_r$ denotes respectively the requested deadline, the requested budget and the request size.

Second and after generating the sub-list(s), the *negoBusiness* computes the best-to-propose offer in each sub-list, denoted by $bestToPropose_i$ where $i$ denotes the *subList* index. The $bestToPropose_i$ is the best offer that the agent can make to the user given a $subList_i$. The best-to-propose offer is calculated using the algorithm 1. From the $subList_1$ the *negoBusiness* chooses the offer having the minimum missing budget (line 1 to 3). From the $subList_2$, the *negoBusiness* chooses the offer having the minimum missing time (line 4 to 6). From the $subList_3$, the provider chooses the closest offer to the consumer request using the *closestToRequest* function (line 7 to 9). This function calculates the distance of each potential offer to the initial request (line 12 to 15). The distance is calculated using the estimated weight of deadline ($EstW_D$) and budget ($EstW_B$). In fact, the provider can guess if the consumer insists on the deadline (for high urgent requests) or insists on the budget (for low urgent requests). Finally, the function returns the offer having the minimum distance to the user request (line 16 to 18).

Finally, the *negoBusiness* triggers a negotiation session with the $negoUser$. If the negotiation succeeds, the request is scheduled via the resource manager.

# 4 Utility and profit-driven decision-making strategies

During the negotiation session, the two agent exchange offers and counter-offers following their decision-making strategies.

## 4.1 The *negoBusiness*'s decision-making strategy

As mentioned in the previous chapter, for the internal context, the *negoBusiness* can use the business provider's goals and the output of the negotiation with the IaaS provider's agent ($negoIaaS$). In our scenario, we assume that a business provider may have one of the following goals: (i) maximize the number of users accepted, with minimum profit (*goal1*); or (ii) maximize the number of users accepted, and maximize the profit (*goal2*). Since we assume that the

**Algorithm 1** Pseudo-code to calculate best-to-propose offer(s)

**Require:** The generated sub-list(s), the user request $(D_r, B_r, RS_r)$
**Ensure:** The best-to-propose offer in each sub-list stored in the list $bestToPropose$

1: **if** $subList_1$ is not empty **then**
2: select $offer_j$ from $subList_1$ with min $fBudget$
3: $bestToPropose_1 = offer_j$
4: **if** $subList_2$ is not empty **then**
5: select $offer_j$ from $subList_2$ with min $fTime$
6: $bestToPropose_2 = offer_j$
7: **if** $subList_3$ is not empty **then**
8: $bestToPropose_3$ = closestToRequest($subList_3$, $(D_r, B_r, RS_r)$)
9: **return** $bestToPropose$

10:
11: **Function** closestToRequest($subList_i$, $(D_r, B_r, RS_r)$)
12: $EstW_D = getEstimatedWeightDealine(D_r, B_r, RS_r)$
13: $EstW_B = 1 - EstW_D$
14: **for** each $offer_p \in subList_i$ **do**
15: $DistanceToRequest_p = EstW_D(D_r - D_{op}/D_r) + EstW_B * (B_r - B_{op}/B_r)$
16: choose $offer_p$ having minimum $DistanceToRequest_p$
17: $bestToPropose_i$=$offer_p$
18: **return** $bestToPropose_i$

resource provisioning is on-demand (i.e no negotiation), so the *negoBusiness* gets information about the price of the VMs and their characteristics from the VM provisioner (and not from the *negoBusiness* consumer role as mentioned in the previous chapter).

### 4.1.1 The *negoBusiness*'s offer evaluation strategy

The request evaluation is different from the evaluation done by the admission control manager. It is based on the "best-to-propose" offers calculated before starting negotiation. In fact, to check if a received request can be scheduled, the negotiation manager compares the request to each of the $bestToPropose_i$ offer where $1 \leq i \leq 3$.
If there is at least one offer $bestToPropose_i$ that satisfies the condition $D_{op} \leq D_r$ and $B_{op} \leq B_r$, then the negotiation manager accepts the $negoUser$ request else the negotiation continues by generating counter-offers.

### 4.1.2 The *negoBusiness*'s offer generation strategy

The offer generation strategy will depend on the the evaluation of the user's preferences. The *negoBusiness* can guess if the user prefers to emphasize deadline over budget or budget over deadline. Indeed, it is intuitive that a request of high urgency occurs for a high budget with a

very tight deadline i.e $w_{deadline} > w_{budget}$. In contrast, a request of low urgency occurs for a low budget with a very relaxed deadline i.e $w_{budget} > w_{deadline}$ [89]. In most cases, the user does not disclose the weights and keeps secret his or her preferences [22], hence the provider cannot know the exact values.

By evaluating the user's preference, the $negoBusiness$ comes closer to meeting the $negoUser$'s initial offer, and so the $negoBusiness$ increases the chance of the $negoUser$ accepting the proposal. If the request is urgent, the consumer preferences on deadline will be greater than the budget, and the consumer may concede on the budget in order to obtain an earlier deadline. However if the request is not urgent, the consumer will want to have a minimum cost, and so the consumer may concede on the deadline in order to obtain a lower budget. The agent $NegoBusiness$ can use one of the following strategies for counter-offer generation depending on the business provider's goal:

1. *pure scheduling-based strategy:*

   In order to maximize the number of accepted requests and to increase the chance of a request being accepted, the provider begins by sending the best-to-propose offer from the sub-list corresponding to the assumed preference of the user. The generation of a counter-offer then comes from one of the other sub-lists. For example, if the request is urgent and the three sub-lists are not empty, the provider proposes $bestToPropose_1$ at the first round, the $bestTopropose_2$ at the second round and the $bestToPropose_3$ at the third round. Otherwise, if the request is not urgent, the provider starts by sending the $bestToPropose_2$, then the $bestToPrpose_1$ and finally the $bestToPrpose_3$. Using this strategy, the negotiation rounds are maximum equal to three (the number of sub-lists).

2. *time-dependent and scheduling-based strategy:*

   While keeping one of the best-to-propose offer as reserved offer, the provider can choose an appropriate negotiation tactic from tactic database to generate the values of the negotiable issues. Given the importance of time factor, The $negoBusiness$ can use a time-dependent concession tactic with scheduling-based strategy to generate counter-offers. In our case, That strategy is used when there is a $bestToPropose$ offer with extra budget (the $subList_2$ is not empty). The negotiation manager keeps the $bestToPropose_2$ as reserved offer and uses the following time-dependent concession function [18] to generate the budget values denoted by $budgetVal$:

   $$budgetVal = min + (1 - \alpha(t)) * (max - min); \tag{IV.7}$$

   where min and max denote respectively the reserved budget value ($B_{op}$ from $bestToPropose_2$) and the maximum budget value (equal to the initial requested budget). Different functions had been proposed for the calculation of $\alpha(t)$ such as polynomial and exponential [18]. We

had chosen the polynomial function for its simplicity:

$$\alpha(t) = k + (1 - k) * (\min(t, t_{max})/t)^{1/\beta} \tag{IV.8}$$

where $k$ denotes the initial concession, $t$ and $t_{max}$ denote the current number of rounds and the maximum number of rounds respectively. The $\beta$ value denotes the convexity degree (i.e concession behavior).

The *negoBusiness* concedes on the issue of budget until reaching the reserved budget (i.e budget leading to the minimum expected profit). By doing so, the provider can avoid disclosing in the first round the budget leading to its minimum expected profit. For the deadline value, the negotiation manager sends at each round $D_{op}$ from the $bestToPropose_2$.

The first strategy is used when the business provider wants to obtain the maximum number of clients with the minimum profit (*goal1*), whereas the second strategy is used when the business provider's primary objective is to maximize profit (*goal2*).

## 4.2 The *negoUser*'s decision-making strategy

The *negoUser* uses the following information communicated by the user as part of its internal context:

- The weights on the deadline and the budget: Each attribute has a weight $w_i$ in the range $[0, 1]$ expressing its importance. For example, for requests of high urgency, users place more importance on the deadline than on the budget. In contrast, for requests of low urgency, users place more importance on the budget than on the deadline.
- The range of each issue, delimited by its worst ($x_{worst}$) and best ($x_{best}$) values
- The preferred value and the reserved value for each issue
- The utility function: The utility value of a negotiable attribute $i$ with value $x$ can be calculated as follows:

  $$U(x_i) = \frac{x_i - x_{worst}}{x_{best} - x_{worst}} \tag{IV.9}$$

  The overall utility of a received offer composed of $n$ attributes is calculated as a weighted sum of each individual utility (IV.9).

  $$U(offer) = \sum_{k=1}^{n} w_i * U(x_i) \tag{IV.10}$$

### 4.2.1 The *negoUser*'s offer evaluation strategy

Given a reserved and preferred value for each issue, the $negoUser$ calculates the reserved and preferred utility using equation (IV.10). The $negoUser$ begins the negotiation by sending the preferred offer, and for each received offer, the $negoUser$ calculates the received utility and accepts an offer if and only if $U(offer_{received}) \geq U(offer_{reserved})$.

### 4.2.2 The $negoUser$'s offer generation strategy

There are different strategies that can be applied to generate a counter-offer. We use a *tradeoff* strategy that consists of increasing the utility of the preferred attribute while decreasing the utility of the other attribute [22]. In other words, if the $negoUser$ concedes on the less preferred issue while increasing the utility of the more preferred issue, then the overall utility does not change.

# 5 Evaluation and analysis

## 5.1 Implementation and experimental settings

We implement our framework using multi-agent system using JAVA and the Java Agent DEvelopment framework (JADE) [5]. Each software agent is negotiating on behalf of either end-user or business provider. Furthermore, business provider's modules are represented by three type of agents representing the admission control manager, the negotiation manager and the resource manager. The agents communicate through the Fipa Interaction Protocol and the negotiation is done through the bilateral version of the *iterated contract net protocol* (the multi-round negotiation protocol) [4]. We simulate different VM types by varying their MIPS. For that purpose, we use the benchmark for MIPS in Cloud VMs [96].
The experiments are carried out for two users' sets:

- Users having requests of low urgency: Those users have relaxed deadlines and low budgets. We consider three types of users' sets: relaxed, medium and demanding, along with minimum initial utility value, relaxed, medium and high, respectively. The users' initial utility values are generated randomly from an interval between minimum initial utility and maximum initial utility. The minimum initial utility value is varied according to the user set and the maximum initial utility is fixed. The values of the experimental parameters associated with those users are presented in Table IV.1. For this set of experiments,

we aim to study the impact of adding negotiation to the classical provisioning (without negotiation).

- Users having requests of high urgency: Those users have soon deadlines and high budgets. The values of the experimental parameters associated with those users are presented in Table IV.2. For this set of experiments, we aim to study the impact of varying the negotiation decision-making strategy of the business provider's negotiation agent ($negoBusiness$).

The values for deadline are generated as in [20]. Given the deadline values, The values for the budget are generated according to the initial utility, which is generally equal to the preferred one. The relation between the values follows the rule the sooner is the deadline requested the higher is the assigned budget.

These experiments were conducted on a laptop with a 64 bit Intel Core 1.8 GHz CPU and 4GB RAM and Windows 7 as operating system. The same environment was used for the other contributions, which will be described in the next Chapters.

**User agent decision-making configuration** The reserved utility is equal to the preferred utility for all users' agents. In other words, the user's agent accepts an offer only if the utility of that offer is equal or greater to his preferred utility. Furthermore, the user's agent follows the tradeOff approach to generate counter-offers during negotiation.
The decision-making configuration described above is the least flexible configuration in negotiation, because the users do not accept concessions. By doing so, we will be sure that for other configurations, our negotiation-based approach will perform better.

TABLE IV.1: Simulation parameters for low urgent requests

| **Parameters** | **value** |
|---|---|
| Request length | $4 * 10^6(MI)$ |
| Number of users | 1000 |
| Max initial utility | $0,9$ |
| Min initial utility (relaxed) | $0,80$ |
| Min initial utility (medium) | $0,82$ |
| Min initial utility (high) | $0,85$ |

## 5.2 Results and analysis

Our objective is to evaluate our negotiation-based provisioning approach and to compare it to a classical provisioning approach *'ProfMinVm'* presented in [20]. For that, we use three performance measurement metrics: (i) the total profit expressing how much the business provider

TABLE IV.2: Simulation parameters for high urgent requests

| **Parameters** | **value** |
|---|---|
| Request length | $2 * 10^6(MI)$ |
| Number of users | 100 |
| Max initial utility | $0,9$ |
| Min initial utility | $0,85$ |
| $t_{deadline}$ | 20 |

profits, (ii) the number of users accepted, and (iii) the average of the received utilities of the accepted users. In what follows, we will first present the experimental results related to the users having requests of low urgency. This first experimentation is conducted in order to compare the classical provisioning to the negotiation-based one. Then, we will present the experimental results related to the users having high urgent requests. The second experimentation is conducted in order to compare the negotiation strategies followed by the business provider's agent.

### 5.2.1 Low urgency requests

To study the impact of adding negotiation, we compare the performance of the provisioning approach without negotiation to the negotiation-based one. The Figure IV.5 shows the different values saved for profit, for the number of users accepted, and for the average utility with respect to the users' initial preferences.

*First observation:*

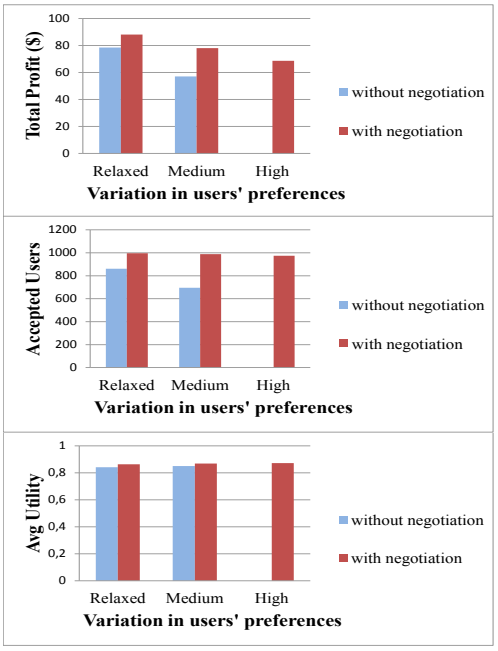


FIGURE IV.5: Algorithms' performance during variation in users' preferences

By increasing the initial utility of users (which is equal to the preferred and reserved one), we note a degradation in profit and the number of accepted users, and an improvement in the average received utility. In fact, the higher the initial utility is, the lower the chance to find a convenient scheduling. If the users are demanding and there is no possible scheduling, the users' requests will be rejected which decrease the total profit. But, for those users, if the request is accepted, the final received utility will be equal or greater than the preferred one which increase the average received utilities. In contrast to the provisioning approach without negotiation that has a large degradation, our algorithm shows little degradation. The degradation in our algorithm is due to the fact that the reserved utility is equal to the preferred utility. Consequently, the users cannot accept an offer just because the received utility is not greater than the initial one, in other words, the users do not accept to make concessions. So, by slightly decreasing the reserved utility (which is the case in the most common negotiation configurations), the degradation for the negotiation-based provisioning approach is improved. In this case, the users accept to make concessions until reaching the reserved utility which increase the chance to reach an agreement.
*Second observation:*
When users are demanding, the performance of the negotiation-based provisioning approach greatly exceeds that of the initial one (without negotiation). The profit and the number of users accepted tend to zero for the initial approach. In fact, the user requests have low budgets, which may even be less than the minimum expected profit. For that reason, those requests will be rejected by the provider. By using negotiation, the provider can propose to increase the budget, thus reaching the minimum expected profit, and proposing a better deadline. So, the overall user utility is not decreased, and the increase in the budget is compensated by a better response time. By harnessing the tradeoff relationship between deadline and budget, the two parties can reach a satisfactory agreement.

In conclusion, for the three sets of users, our algorithm's performance exceeds that of the initial one in terms of profit, number of accepted users, and average received utility.

### 5.2.2 High urgency requests

To study the impact of varying the business provider decision-making strategy, we compare the pure scheduling-based strategy (strategy 1) where the business agent (*negoBusiness*) sends the best-to-propose offer at each round, to the combined one called time-dependent scheduling-based strategy (strategy 2) where the *negoBusiness* uses a time-dependent concession tactic within the scheduling-based strategy. Figure IV.6 shows the different values saved for profit, for the number of users accepted, and for the average utility with respect to the strategy used (without negotiation, nego-strategy 1, nego-strategy 2)

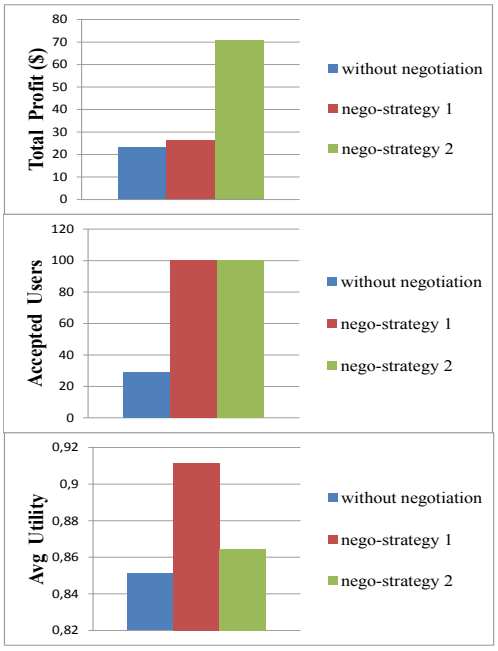


FIGURE IV.6: Algorithm performance during variation in negotiation strategy

The total profit using strategy 1 exceeds slightly the total profit using the strategy without negotiation. This is because the provider using strategy 1 generates offers based on the minimum expected profit. The provider can adopt this strategy to have the maximum number of clients even with little profit. We note that the total profit using strategy 2 exceeds the the total profit using strategy 1. In fact, the provider using strategy 2 (time-dependent and scheduling-based strategy) is doing little concessions during negotiation. In fact, the provider starts by proposing the preferred value of the budget (generally high budget), and concedes to the reserved value (minimum expected profit) only when negotiation deadline approaches.

We note that the number of accepted users using strategy 1 and strategy 2 exceeds highly that of the strategy without negotiation. In fact, without negotiation, the number of rejected users can be increased mainly when the users are demanding. The number of accepted users using the strategy 1 and the strategy 2 are equals. In fact, regardless of the offers received from the provider, the users generate counter-offers using the tradeoff strategy until reaching the negotiation deadline. And since the maximum number of rounds (negotiation deadline) in our configuration is relatively high, the negotiation ends with success using the two strategies. However, the number of rounds needed to reach the agreement using the strategy 2 is greater than strategy 1.

We note that the average utilities is higher using the strategy 1 than the strategy 2. In fact, the provider following the strategy 1 is proposing good prices (based a the minimum expected profit), which lead to increasing the users satisfaction. However, the provider following the

strategy 2 focus on maximizing the profit by proposing high prices, which lead to a decreased user satisfaction.

# 6 Conclusion

In this chapter, we have proposed a bilateral negotiation approach for an efficient SaaS provisioning. The negotiation between business provider and end-user aims to optimize the provider's profit and maximize customer satisfaction. We base our approach on an existent provisioning approach, where the users' requests are rejected if the business provider cannot schedule the request. We propose to bring more flexibility to the provisioning approach. Instead of rejecting the users' request, the negotiation agent acting on behalf the business provider (*negoBusiness*) tries to propose alternative schedules to the user agent. To do so, the *negoBusiness* communicates with the provisioning modules (admission control, scheduler, VM provisioner) in order to find alternative schedules satisfying both parties. We have propose two decision-making strategies for the *negoBusiness*, those strategies could be used according to the business provider's goals. The experimental evaluation demonstrates the benefit of including negotiation in the provisioning process. In the next chapter, we propose to extend the negotiation approach to be adaptive, considering dynamic changes in workload.

# Chapter V

# Adaptive and concurrent negotiation for an efficient SaaS application provisioning

Contents

# 1 Introduction

Business providers offer highly scalable applications to end-users. To run the users' requests efficiently, business providers must take the right decision about requests placement on virtual resources. The business providers aim to optimize their profit and satisfy users. So, an efficient scheduling decision becomes a challenging task due to finite resources at a time. Negotiation-based approaches are promising solutions when dealing with conflicts. Using negotiation, the users and providers may find a satisfactory schedule. However, reaching a compromise between the two parties is a cumbersome task due to workload constraints at negotiation time. In fact, the users may have specific requirements with limited budget, that the provider could not satisfy due to workload status at point of time $t$. The workload status is defined by the set of Virtual Machines (VMs), VMs state and characteristics, and the requests affected to each VM. The gap between the user's requirements and the provider's capability may lead to difficulty to reach a compromise. Negotiation session are generally limited by time (which is called negotiation deadline), it is obvious since the negotiation cannot be infinite. If the two parties does not reach an agreement before the negotiation deadline, the negotiation ends with failure.

Most of negotiation approaches designed in the Cloud are mono-session. Those approaches enables exchange of offers and counter offers (i.e multi round) only during the negotiation session which is limited by the negotiation deadline. Furthermore, the negotiation decision-making strategies are based on the situation at negotiation time which may not allow reaching an agreement. If the negotiation fails, the users' requests are rejected. So, we face a situation that not only the users are not satisfied but also the business provider may loss in profit because of cumulative loss of clients.

Given the dynamicity of the cloud (variation of workload, dynamic resource pricing, etc.), at point of time $t+1$, there may be new elements such as new idle resource and new resource capacity which may change the previous negotiation outcome. We propose to harness those changes by adapting the negotiation behavior accordingly. In this chapter, we propose an adaptive negotiation approach that considers the workload change and keeps negotiating concurrently with non-accepted users according to those changes. So, the provider can accept more users which will lead to an increased profit and better satisfaction of end-users. To do so, when the negotiation fails, the business provider gives another chances to non-accepted users by renegotiating simultaneously with them when there is a change in the elements considered in the first negotiation. Those elements change are given by the others business provider's provisioning modules (VM provisioner and scheduler). The experiments show that the adaptive and concurrent negotiation improves the business provider profit, the number of accepted users' requests, and the client satisfaction, as compared with the bilateral negotiation.

This chapter is organized as follows. In section 2 we give an overview of the adaptive negotiation framework for SaaS provisioning. In section 3 we detail the interactions between business provider's provisioning modules during the provisioning process. The renegotiation decision-making strategies are detailed in section 4. Section 5 presents experiments to assess the adaptive negotiation approach for SaaS provisioning. Finally, in Section 6, we conclude the chapter.

# 2 Overview of the adaptive negotiation framework for SaaS provisioning

The adaptive negotiation framework is based on the proposed multi-layer negotiation framework(see Figure III.4) and the compute-intensive SaaS provisioning scenario described in the previous Chapter. The adaptive negotiation is an extension of the bilateral negotiation proposed in the previous Chapter IV.

## 2.1 Scenario description

The compute-intensive application provisioning scenario is detailed in the previous chapter IV. The application request/offer contains essentially the following attributes: the deadline, the budget allocated by the user for request processing, the request length, and the penalty rate that represents how much the business provider would pay as compensation in case of violation. We assume that the deadline and the budget are negotiable issues.
As mentioned earlier, the application provisioning depends on resource provisioning. In fact, to execute the users' requests, the business provider needs resources that may be owned by different Infrastructure-as-a-Service (IaaS) providers. We assume the VM provsioning is on-demand and the VM Billing is per Time Unit (BTU), as for example, with the Amazon on-demand instances for which the unit is equal to one hour [64].

## 2.2 The proposed framework

The Figure V.1 illustrates the adaptive negotiation framework for SaaS application provisioning. The upper layer is composed of the negotiation agents ($negoUser$) acting on behalf users. The $negoUser$ is able to carry out automated negotiation with the business provider's negotiation agent ($negoBusiness$) according to the user's preferences and requirements. For the negotiation-based SaaS provisioning, The business provider uses the following modules:

- Admission control manager is responsible for request mapping and evaluation. To be accepted, the request must verify both the feasibility and the profitability conditions. In

other words, the available resources must ensure that the request processing will terminate before the requested deadline. Furthermore, the request processing must give some profit to the business provider. In classical provisioning, if one of the two condition is not verified the user's request is rejected [20]. In the previous Chapter IV, we have explained how we enhance the classical admission control module in order to support negotiation. The idea was to find alternative schedules instead of rejecting the request. If the request does not satisfy the two conditions, the admission control manager forwards the request with the alternative schedules to the *negoBusiness* agent.

- Schedule manager: is responsible for assigning users' requests to available time slots. It has information about the load of each owned resource.
- VM provisioner: is responsible for VM provisioning and deploying application instances on that VM. It has information about resources for lease (BTU price, initiation time, provider owner, capacity, etc.).

Based on the communication with the above mentioned provisioning modules, the *negoBusiness* is able to carry out two types of automated negotiation depending on the situation. The first type is the **bilateral negotiation** which happens when the *negoBusiness* receives a request which cannot be accepted from the admission control manager. In this case, the *negoBusiness* opens a bilateral negotiation session with the *negouser*. The offers generated during the negotiation are based on information given by the admission control manager. The details of the bilateral negotiation approach can be found in the previous chapter IV. In that approach, as in the most current negotiation approaches, if the negotiation fails, then the user's request is rejected. The negotiation fails due to a low user's budget and/or not enough resource capacity to finish the user's request before the requested deadline.
In this chapter, we propose to enhance the bilateral negotiation in order to take into account the change in elements considered in the first negotiation. In other words, the *negoBusiness* should adapt his behavior according to the changes. Here, we assume that the elements change consists of the availability of new resource capacity. This change can be achieved either via the instantiation of new VMs or extending the lease period of an already existing VM. The considered elements changes are denoted by workload changes. Those changes are triggered either by the VM provisioner or the schedule manager. The second type of negotiation that can be handled by the *negoBusiness* is the **adaptive negotiation** according to workload changes.

The key idea of the adaptive negotiation approach is that, if the first negotiation fails, the *negoBusiness* stores some information about the users in the *renegotiation list* in order to renegotiate with them later (when there is a change in workload that may enable the acceptance of their requests). When there is a change in workload, the *negoBusiness* opens a renegotiation session with the non-accepted users during the first negotiation (from the *renegotiation list*).

During the renegotiation session, the *negoBusiness* negotiates simultaneously with the potential interested users by the workload changes. The renegotiation decision-making strategy followed by the *negoBusiness* aims to optimize the use of the new resource capacity.

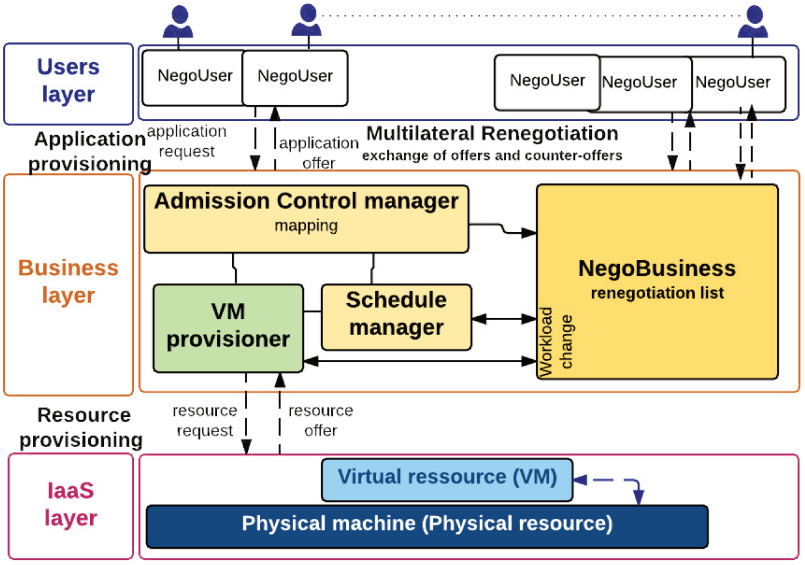


FIGURE V.1: Adaptive negotiation framework for SaaS provisioning

In what follows, we give more details about the interaction between the *negoBusiness* and the others provisioning modules for an adaptive negotiation. Then, we describe the renegotiation decision-making strategy followed by the *negoBusiness* aiming to enhance the resource utilization and customer satisfaction.

## 3 Adaptive negotiation-based SaaS provisioning process

Figure V.2 shows the interaction between the admission control manager, the *negoBusiness* and the resource manager for an adaptive negotiation-based provisioning. The resource manager represents both the VM provisioner and the Schedule manager.

As mentioned earlier, our approach is an extension of the bilateral negotiation (first type of negotiation). The extensions we made are presented in Grey boxes, while white boxes present the already existing modules. The process starts by evaluating the user's request by the admission control manager. The admission control process is detailed in the previous chapter. If the application request is acceptable (satisfy both the deadline and the cost condition), the admission

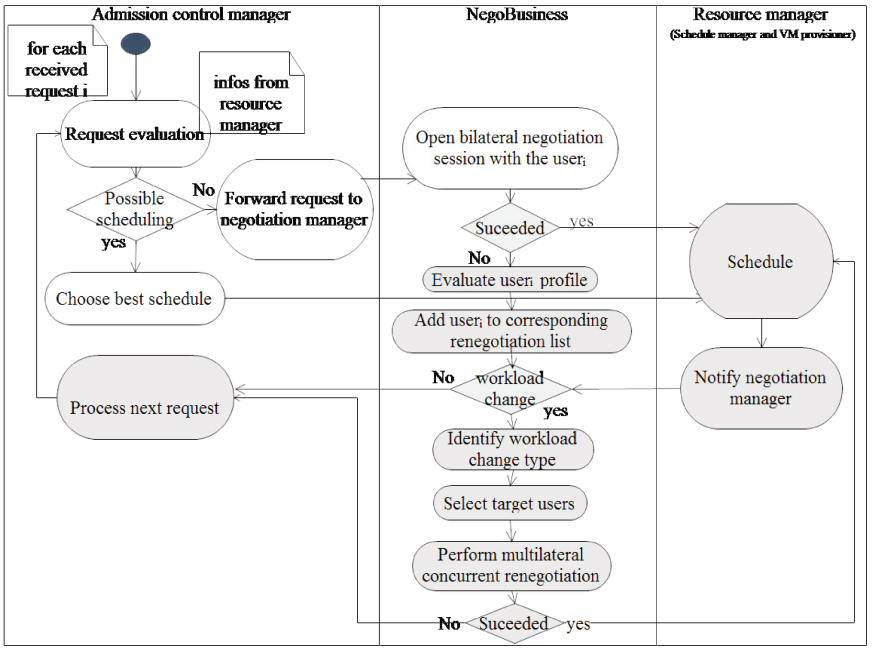


FIGURE V.2: Interaction between the *negoBusiness* and the business provider's provisioning modules for an adaptive negotiation-based SaaS provisioning

control manager accepts the request. If one of the two conditions is not satisfied, the admission control manager forwards the request to the *NegoBusiness* in order to launch a bilateral negotiation.

If the bilateral negotiation fails, the *negoBusiness* evaluates the user's profile in terms of needs and preferences. The user's profile indicates why the negotiation did not succeed especially with those users. In fact, each rejected user's request has specific requirements that the provider could not satisfy due to the workload status at time point $t$. At time $t+1$, there may be a variation in workload (due to a newly idle resource, or a new resource capacity) that may change the first evaluation. In the context of compute-intensive application provisioning, we can define two types of user's profile according to the reason of negotiation failure:

- The user's request can be executed by an existing VM (i.e among instantiated VMs, there is at least one VM that can execute the request) and the business provider needs to extend the lease period of that VMs (to include a new BTU). But, the user's budget cannot cover the BTU cost. Here the negotiation did not succeed because there is no idle resources for which the provider will not pay extra cost. If at time $t+1$, the provider will have idle resource, the request could be accepted.
- The user's request cannot be executed by any of the existing VM(s) and the business provider need to lease a new VM. For example a VM with high capacity is needed in order to meet the requested deadline. But, the user's budget cannot cover the cost of that VM instantiation. Here the negotiation did not succeed because there is no exiting

resources with the needed capacity (i.e the existing resources cannot fulfill the request's requirements). If at time $t+1$, the provider will have a new VM with the needed capacity, the request could be accepted.

The two types of profile cited above deals with users having a low budget and a high reserved utility value (i.e., demanding users). That value is included in the user's preferences and represents the minimum degree of satisfaction under which the user cannot accept the offer. Generally those users have low-urgent requests, because users with high-urgent requests have high budget that can cover the cost of extending the lease period or even the cost of the instantiation of new VMs. Furthermore, given the requested soon deadline, the users of high-urgent requests do not have time for a renegotiation after a period of time has elapsed.

After evaluating the user profile, the corresponding $negoUser$ is added to the appropriate renegotiation list. We define a renegotiation list for each user profile. When there is a change in workload, the resource manger notifies the $negoBusiness$ and send the nature of change. For example, when there is a new request accepted, its scheduling results either on a new VM intantiation or the extend of lease period of an already existing VM.

The negotiation manager then selects the appropriate renegotiation list(s) and triggers a renegotiation with the users in this list. When the lease period of an existing VM is extended, the $negoBusiness$ renegotiates with the first type of users. Although, when a new VM is instantiated, the $negoBusiness$ renegotiates with the two types of users.

Note that in our approach, the communication between the admission control manager and the $negoBusiness$ should be synchronized via message exchange. The admission manager does not process new requests while the $negoBusiness$ is still negotiating. At the end of the renegotiation session, if there are newly accepted users (with whom the renegotiation succeeded), the $negoBusiness$ informs the resource manager to schedule those requests. Otherwise, the $negoBusiness$ sends a notification to the admission manager in order to process further requests.

More details about the renegotiation session and the decision-making strategies followed by the $negoBusiness$ and the $negoUser$ are presented is the next section.

## 4 Renegotiation decision-making strategies

In this section, we present how the $negoBusiness$ renegotiates concurrently with the $negoUser$(s) with whom the first negotiation fails. First, we describe the $negoBusiness$'s strategy that guides the offer's evaluation and generation during the renegotiation session. Second, we present some strategies that the $negoUser$ could follow.

## 4.1 The *negoBusiness*'s decision-making strategy

Algorithm 2 details the provider's renegotiation strategy. It takes as input three parameters: the list of $negoUser$(s) with whom the first negotiation fails ($renegList$), the maximum number of rounds during the renegotiation $nbrMax$ and the new workload status $W$ which can be defined as $(vm_{ID}, est)$ where the $vm_{ID}$ denotes the $ID$ of the newly available VM (newly instantiated or newly idle resource time). The $est$ denotes the earliest start time where the VM cannot start execute the request before $est$. The output of the algorithm is the renegotiation outcome, which contains the achieved agreement(s).

**Algorithm 2** renegotiation strategy

**Require:** $renegList$, $nbrMax$, $W$
**Ensure:** The achieved agreement(s)
1: **if** $renegList \neq \emptyset$ & $nbrRound \leq nbrMax$ **then**
2: **for** each $user_j \in renegList$ **do**
3: $compT_j$=EvalCompT($ReqLength_j$,$W$)
4: $tradeOff_j = (compT_j - deadReq_j)/deadReq_j$
5: $budget_j = budgReq_j + tradeOff_j * budgReq_j$
6: $offer_j$=GenerateOffer($budget_j$,$compT_j$)
7: SEND $offer_j$ to $user_j$
8: $nbrRound$++
9: WAIT FOR RESPONSES(x)
10: **if** $responses \neq \emptyset$ **then**
11: $accList$=EvalResp($responses$)
12: **if** $accList \neq \emptyset$ **then**
13: $user_{acc}$=ChooseBest($accList$,$W$)
14: SEND ACCEPT to $user_{acc}$
15: update $W$
16: remove $user_{acc}$ from $renegList$
17: $renegList$=$accList$
18: GO to line 1
19: **else**
20: exit
21: **else**
22: exit
23: **else**
24: Exit

The *negoBusiness* starts the renegotiation session by preparing a customized offer to each $negoUser$ in the $renegList$, assuming that the number of the current round ($nbrRound$) does not exceed the $nbrMax$ (line 1 to 7). To prepare an offer, the algorithm calculates the estimated completion time of the user's request ($compT_j$) by using the $EvalCompT$ method. This method takes as input the user's request length and the new workload status $W$ (line 3). Then, the degree of tradeoff ($tradeOff$) is calculated. It measures how much the provider will enhance or lower the deadline value compared to the requested deadline $deadReq$ (line 4). Then, the

*negoBusiness* calculates the budget that it will propose (*budget*) using the tradeoffs between the deadline requested *deadReq* and the budget requested *budgReq* (line 5). The better the completion time that is proposed, the higher the budget that is proposed, and vice versa. Finally, the *negoBusiness* prepares the offer to be sent, which contains the estimated completion time and the calculated budget (line 6). After sending to each $negoUser$ the customized offer (line 7), the *negoBusiness* increments the number of rounds (line 8) and waits for responses for a predefined time $x$ (line 9). If no response is received (line 10), then the renegotiation session is closed. Otherwise, the *negoBusiness* evaluates the responses containing counter-offers using the *EvalResp* method. This method has as output the list of acceptable received offers from $negoUsers$ denoted by *accList* (line 11). In fact, the $negoUser$(s) may propose counter-offer(s) that do not satisfy the cost and the deadline condition even when considering the workload change. If the acceptable list is empty (line 12) (no acceptable counter-offer received from $negoUser$), the renegotiation session is closed. Otherwise, the *negoBusiness* chooses and the $negoUser$ whose request leads to the best profit using *ChooseBest* method (line 13 and 14). This method calculates the profit for each $negoUser$'s request in *accList* using the following formula:

$$profit_i = budget_i - cost_i(BTU, est, compT_i), \tag{V.1}$$

where the $budget_i$ denotes the budget proposed by the user $i$ in the counter-offer. The $cost_i$ denotes the resource cost and can be calculated given the $BTU$ of the VM where to execute the request ($vm_{ID}$). The $est$ and $compT_i$ denote the start time and the completion time of the request, respectively.

After choosing the best $negoUser$, the workload status is updated to include the newly accepted request (line 15) and the accepted user is removed from the $renegList$ (line 16). The *negoBusiness* could continue renegotiating with the users in *accList*, until there is no acceptable user's request or until the number of rounds is equal to $nbrMax$ (line 18).

## 4.2 The *negoUser*'s decision-making strategies

The decision-making strategy followed by the $negoUser$ is different from provider's one. It is based on users' preferences expressed in terms of a utility function. The utility function measures the degree of satisfaction of a given offer. Generally, the utility value of a given offer results from a weighted sum of the utilities for the individual attributes. In particular, in multi-issue negotiation, the user assigns a weight to each issue, expressing the importance of such issue.
The strategy followed by the $negoUser$ is the same as in the previous chapter 4: before starting the negotiation, the user defines a reserved utility value, and the $negoUser$ accepts an offer only if the utility of the offer received is greater then the reserved utility value.

For counter-offer generation, the *negoUser* use tradeOff approach which consists of increasing the utility of the preferred attribute while decreasing the utility of the other one.

# 5 Evaluation and analysis

## 5.1 Implementation and experimental settings

We implemented the adaptive negotiation framework V.1 using Multi-agent system. The details about the implementation are given in the evaluation Section of the previous Chapter IV. We have extended the behavior of the agents in order to support adaptive negotiation. Whenever there is a change in workload, the agent representing the resource manager sends ACL message to the negotiation agent *negoBusiness*. Furthermore, the negotiation agent use the one-to-many version of the *iterated contract net protocol* to renegotiate with user's agents.
As mentioned in Section 3, the proposed approach is suitable for users having requests of low urgency (low budget and relaxed deadline). So, we assume that the *negoBusiness* renegotiates only with those *negoUser*(s) (negotiating on behalf the users). The users having requests with high urgency (high budget and soon deadline) are generated to create variation in workload. The initial preference and initial request of users are generated randomly so as to have an initial utility greater than 0.8 (demanding users). The user agents follow the strategy described in 4.2 with the assumption that the reserved utility value is equal to the initial one.
We measure performance using three metrics: 1) the total profit calculated as the sum of accepted users' budgets minus the total resource costs and the penalty costs (if any), 2) the number of users accepted, and 3) the average of the received utilities of accepted users. To evaluate our proposed approach, we conduct two series of experiments. For the first one, we calculate the total profit and the number of accepted users while varying the number of users generated. For the second one, we calculate the average received utilities for 100 users generated.

## 5.2 Results and analysis

We evaluate the two versions of our proposed approach obtained by varying the *negoBusiness* agent's renegotiation strategy (1) the first version noted "adapt R1" uses one-round concurrent renegotiation session, (2) the second version noted "adapt Rx" uses a multi-round concurrent renegotiation session. The maximum number of rounds is defined dynamically according to the renegotiation session stop condition. In our case, the renegotiation session ends when all users reject the offer or there are no more users in the renegotiation list. We compare the proposed approach to the following ones:

1. classical provisioning approach (without negotiation) noted "no nego". The approach uses take-it-or-leave-it strategy and it is used by the current cloud providers. We have chosen the algorithm presented in [20] because it performs more efficiently when evaluated and compared with the reference scheduling algorithms. More details about that choice is given the previous Chapter.

2. bilateral negotiation-based provisioning (non-adaptive negotiation) noted "no adapt". The bilateral negotiation approach is an extension of the classical provisioning described above. This approach is based on a mono-session negotiation between the $negoUser$ and the $negoBusiness$ (i.e without renegotiation). The $negoBusiness$ negotiates with the $negoUser$ when it is not possible to schedule the user's request. When, the negotiation fails, the user's request is rejected.

Concerning the analysis of the results, we are interested in the difference between the values generated for each approach and not the values in themselves.

**First observation** Figure V.3 shows the different values saved for the total profit and the number of accepted users for each approach with respect to the number of generated users.

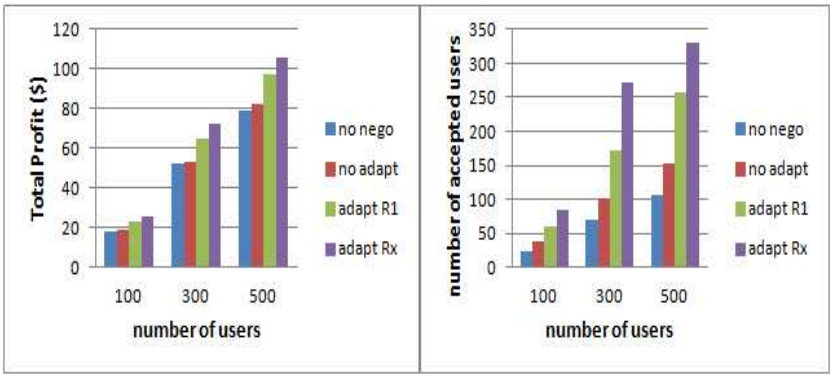


FIGURE V.3: The total profit and the number of accepted users

We observe that the total profit and the number of accepted users is improved when using adaptive negotiation ("adapt R1", "adapt Rx") regardless of the total number of generated users. This is because, the adaptive negotiation increases the chance of a request being accepted by considering potential opportunities (workload change). When the number of rejected users decreases, the total profit systematically increases. In fact, our renegotiation strategy aims to maximize the number of accepted users while minimizing the cost of rented resources, so the accepted users give the provider more profit without paying high costs. Furthermore, we note that total profit and the number of accepted users increase when the number of generated users increases. It is obvious that having more users leads to an increased number of accepted users and an increased profit. When comparing the two versions of the adaptive negotiation-based

approaches, the performance of "adapt Rx" exceeds the "adapt R1". This is expected, since the multi-round renegotiation increases the chance to converge towards an agreement compared to the one-round renegotiation. Thus, the number of accepted users will be improved leading to a higher profit. Note that, the improvement is due to users having low urgent requests, because we assume that the negotiation manager renegotiates only with those users.

**Second observation** Figure V.4 shows the different values obtained for the average received utility. We observe that the customer satisfaction is enhanced using adaptive negotiation and mainly using the multi-round renegotiation approach "adapt Rx". This is due to the design of the renegotiation decision-making strategies. In fact, the user accepts an offer only if the received utility is greater than the reserved one (which is equal to the initial one). Also, the provider's offer-generation strategy is based on the tradeoffs between the attributes. The higher is the budget proposed, the better is the completion time proposed and vice versa.

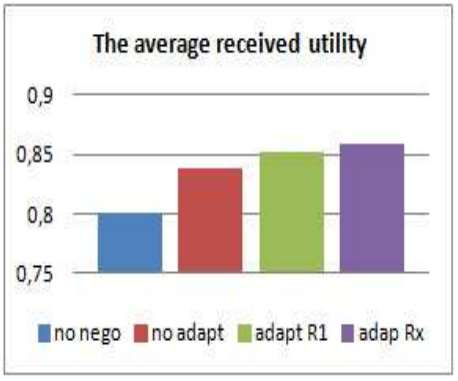


FIGURE V.4: The average received utility

As conclusion, the adaptive negotiation-based approach using multi-round renegotiation provides a win-win framework for users having requests of low urgency and business providers. In fact, the proposed approach performs better than the other ones in terms of business provider's profit, number of accepted users and customer satisfaction.

# 6 Conclusion

In this chapter, we have proposed an adaptive negotiation approach according to workload changes for SaaS provisioning. Due to the gap between the users' requirements and the business provider's capabilities at negotiation time, the negotiation may not reach a compromise. By considering the workload changes, the business agent *negoBusiness* could renegotiate with users' agents with whom the fist negotiation has been fail. The renegotiation decision making strategy followed by the *negoBusiness* aims to maximize resource utilization and also customer

satisfaction. The experimental evaluation demonstrates the benefit of the adaptive negotiation in the provisioning process. In the next chapter, we consider an other type of renegotiation triggered by an unexpected event that may cause the violation of an already established SLA.

# Chapter VI

# Proactive renegotiation for an efficient SaaS application provisioning

Contents

# 1 Introduction

Automated negotiation in the Cloud is primarily used to establish an SLA between clients and providers. It happens generally in the first phase of the service-provisioning process (before the SLA establishment itself). In the first phase, in order to maximize the number of clients and minimize the costs of renting sufficient computer resources, the business provider adopts a profit- and SLA-aware scheduling algorithm. The schedule must guarantee that the SLA is met, while also maximizing the profit.

Given the highly dynamic nature of the cloud environment, *unexpected events* may affect the initial scheduling plans, which leads to unanticipated SLA violations. Thus, an unaccounted event may create a lose-lose situation between provider and client. If the SLA is violated the provider must pay the potentially high penalty that is negotiated within the original SLA. But from the client's viewpoint, an SLA violation may cause cancellation of a business-critical job, and no ordinary SLA penalty can compensate for the loss of the client's business. The provider's reputation could also suffer as the number of such SLA violations grows, resulting in loss of future clients. For these reasons, the provider and the client would normally prefer to renegotiate using a new SLA with a new deadline (i.e., an extension beyond the first deadline), rather than pay a steep penalty and accept the cancellation of a business-critical job.

Most of the literature assumes that once an SLA is established, it cannot be renegotiated [23–26]. The concept of *renegotiation* has not yet been well studied [27]. For the best of our knowledge, there is no concrete renegotiation decision-making strategy which had been proposed for cloud application provisioning.

In this chapter, we propose an automated renegotiation-based approach when detecting an unexpected event during the SaaS provisioning process. In our approach, the business provider proactively renegotiates with the clients whose jobs may be in violation of the SLAs, in order to minimize the loss in profit and in order to assure the continuity of the service. The renegotiation approach is composed of two steps. The first step happens when the business provider detects an unexpected event. To avoid an SLA violation, the business provider may take rescheduling actions. We consider different alternative rescheduling options for the business provider. The first step consists of the selection of an option for profit-aware rescheduling which leads to a minimum loss in profit while also minimizing the number of canceled jobs. At the second step, the business provider triggers a renegotiation with those end users whose jobs may terminate after deadline according to the rescheduling option selected at the first step. Experiments show that this new approach minimizes the loss in profit of the business provider and minimizes the number of cancelled jobs, as compared with enforcing the original SLA.

This chapter is organized as follows. Section 2 presents an overview of the renegotiation framework showing the main modules considered for proactive renegotiation. Section 3 and Section 4 describe the first step and second step, respectively, of the renegotiation-based SaaS provisioning process. Section 5 presents experiments to assess the renegotiation-based approach. Conclusions are presented in Section 6.

## 2 Overview of the renegotiation framework for SaaS provisioning

The proposed framework as shown in VI.1 is an extension of the framework presented in the previous chapters. For the business providers' provisioning modules, in addition to the admission control manager, the schedule manager and the VM provisioner (which had been described in the previous chapters), we consider the monitor module which is able to detect unexpected event prior to SLA violation. Furthermore, we extend the behavior of the *negoBusiness* agent in order to support proactive renegotiation by communicating with the monitor and the others provisioning modules. In our scenario, we suppose that the SLA is established and the the schedule is already defined according to that SLA.

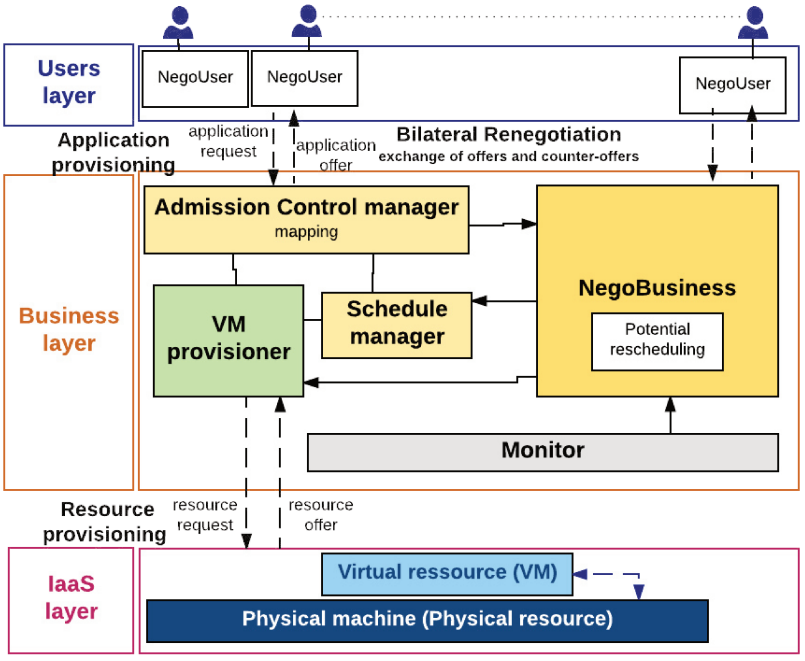


FIGURE VI.1: Renegotiation framework for SaaS provisioning

After signing the contract, *unexpected events* may later occur that can impact the current scheduling. To avoid an SLA violation, the *negoBusiness* tries first to take rescheduling actions via the communication with the Schedule manager and the VM provisioner. However, maybe no rescheduling can meet the previously signed SLA. For example, migrating a job to another VM after failure may delay the completion time beyond the agreed upon deadline. The SLA model generally assumes that once the deadline is violated, the job is automatically cancelled and a penalty is paid. This is the scenario in which automated renegotiation becomes important, as part of a second phase of service provisioning. In that phase the *negoBusiness* invoke a new renegotiation session with the $negoUser$ (acting on behalf end-user), in order to minimize the SLA penalty costs and in order to ensure the continuity of service. The renegotiation decision-making strategies are based on the first step where the *negoBusiness* chooses a rescheduling option given potential scheduling alternatives. The option selected must lead to minimum loss in profit while also minimizing the number of cancelled jobs.
In what follows, we detail how automated renegotiation can be used to handle unexpected events, as part of a second phase in SaaS application provisioning.

# 3 Selection of an Option for Profit-aware Rescheduling

When detecting an unexpected event that alters the initial scheduling, the *negoBusiness* takes rescheduling actions in order to avoid SLA violations to the extent possible. Generally, the provider may have more than one rescheduling option. For this reason, we propose an algorithm for the selection of an option for profit-aware rescheduling. In this section, we model first the unexpected event and the rescheduling option. Then, we present our algorithm for selection of a rescheduling option.

## 3.1 Definition of an Unexpected Event

An unexpected event leads to a change in the situation under which the already signed SLAs had originally been negotiated. Indeed, the schedule contracted by the business provider in the first phase may be affected, thereby leading to violation of the original SLA. The unexpected events can be classified into two categories: (1) *resource events*, for example VM failure, failure of the currently executing job, etc. The jobs scheduled on that VM may be affected, and so the initial scheduling may be altered. (2) *business events*, such as a new incoming client needing immediate execution with no additional VMs available from the IaaS provider. Thus, the business provider may choose to execute a new job on an already active VM even though there exist prior jobs (either running, or scheduled but not yet started). An unexpected event can be specified using two parameters: the time at which the event occurs, $t_{event}$; and the set of resources affected by

the event, $vm_{ID}$. As mentioned earlier, the unexpected event is assumed to be detected just prior to SLA violation through a monitoring module.

## 3.2 Definition of the Rescheduling Option

The rescheduling option is composed of potential scheduling actions applied to accepted and scheduled jobs (which may either be running or not yet started). Two examples of rescheduling actions are: (i) the job may be proactively migrated a new computer; where the job can be either restarted from a previous checkpoint image (if the provider is invoking periodic checkpointing to protect against catastrophic failure) or restarted from the beginning, (ii) the job may be postponed until the resource where it is scheduled will be available, (iii) the job may be inserted in already existing resource. A scheduling action defines when and where to place a job. A scheduling action $Ac$, applied to a job $j$, can be defined as:
$(type, j, estimated\_start\_time, vm_{ID}, compT, actionCost)$
where: the *type* denotes the scheduling action type. For example: migrate, insert, postpone. The $estimated\_start\_time$ and $vm_{ID}$ define when and where to start the job, respectively. The $compT$ denotes the estimated completion time, and can be calculated based on the information given by scheduling action. The $actionCost$ denotes extra costs due to the rescheduling action taken.

Hence, a rescheduling option, denoted $Op$, is defined as follows: $Op = \{Ac_j\}$, where $j \in$ {rescheduled_jobs}. Each rescheduling option has as output a list of rescheduled jobs ($resch\_List$) and the list's rescheduling information given by the scheduling action.

## 3.3 Algorithm for Selection of a Profit-aware Rescheduling Option

For the selection of a rescheduling option, we consider two metrics: 1) the $lossInProfit_p$, which calculates the SaaS provider loss in profit when choosing rescheduling option $p$; and 2) the number of potential cancelled jobs when choosing option $p$, denoted by $nbrJobs_p$.

1. *The $lossInProfit_p$ for the provider:* This include two parameters: a) The $actionCost$ define the cost due to the action. For example, if the action is to migrate the job to a new $VM$, the cost of the action will be equal to the price of provisioning a new $VM$. b) The $penaltyCost$, defined as the SLA violation cost, which can be calculated for a job $j$ using

the following formula:

$$penaltyCost_j = \begin{cases} 0, \text{if } compT_j \leq respT_j \\ pr_j * delay_j, \text{if } respT_j \leq compT_j \leq dl_j \\ fixedPenalty_j, \text{if } dl_j < compT_j \end{cases} \quad \text{(VI.1)}$$

where $respT_j$ is the agreed upon *response time*, and $dl_j$ is the agreed upon *deadline*. The value $pr_j$ indicates the *penalty rate*. The $fixedPenalty_j$ denotes the penalty paid in case of violation.

2. *The number of potential cancelled jobs, $nbrJobs_p$:* This calculates the jobs whose estimated completion time $compT_j$ are greater than the deadline $dl_j$ of the initial SLA.

The proposed algorithm, Algorithm 3, below, takes as input the list of possible rescheduling options that the *negoBusiness* can choose after receiving the notification of the unexpected event. The algorithm returns a scheduling option and associated rescheduling information for each job such that the number of cancelled jobs is minimized and the loss in profit is minimized.

**For each possible rescheduling option**: First, Algorithm 3, below, calculates the estimated completion time $compT_j$ for each job $j$, based on the action $Ac_j$ applied to this job (line 5). Second, the algorithm calculates the $lossInProfit$ as a sum of the loss in profit for each of the rescheduled jobs (lines 6 and 7). Third, the algorithm selects the potential cancelled jobs whose *compT* are greater than the agreed upon deadline, calculates *nbrJobs*, and stores the rescheduling information for those jobs in *potentialCancelJobs* (line 8 to 10). Then, the algorithm stores the option results in *optionsResults* (line 13). Finally, the algorithm selects the option noted $optionsResults_s$ having the minimum *nbrJobs* and the minimum $lossInProfit$ (line 14) using the *selectBestOption* function. We propose to use utility functions in order to select the most convenient option. **For each possible rescheduling option**: the *selectBestOption* function calculates the utility values *UtLoss* and *UtNbrJobs* for $lossInProfit$ and *nbrJobs* using equation IV.9 (line 21 and 22). The worst and best values for the $lossInProfit$ are the maximum and minimum $lossInProfit$ values selected from the *optionResults*, respectively. Likewise, the worst and best values for *nbrJobs* are the maximum and minimum *nbrJobs* values selected from the *optionResults*, respectively. Then, the function calculates the option's distance to the best option (line 23). The best option has $UtLoss = 1$ and $UtNbrJobs = 1$, because the maximum value of the utility is equal to one. Finally, the function returns the option having the minimum distance to the best option (line 24 to 27). The *negoBusiness* will renegotiate based on the results of the rescheduling option selected ($optionsResults_s$)

**Algorithm 3** Pseudo-code for selection of rescheduling option

**Require:** The list of possible rescheduling options
**Ensure:** The rescheduling option leading to the min $lossInProfit$ and min $nbrJobs$

**for** each $p \in$ possible rescheduling options **do**
  $lossInProfit_p = 0$
  $nbrJobs_p = 0$
  **for** each $j \in resch\_List_p$ **do**
    $compT_j = getCompT(Ac_j)$
    $lossInProfit_j = penaltyCost(compT_j) + actionCost(Ac_j)$
    $lossInProfit_p = lossInProfit_p + lossInProfit_j$
    **if** $compT_j > dl_j$ **then**
      $nbrJobs_p++$
      Add resch info from $resch\_List_j$ to $potentialCancelJobs_p$
    **else**
      **continue**
  Store $lossInProfit_p$,$nbrJobs_p$,$potentialCancelJobs_p$ in $OptionsResults_p$
$OptionsResults_s$=selectBestOption($OptionsResults$)
**return** $OptionsResults_s$

**Function** selectBestOption($OptionsResults$)
$minDistance = \sqrt{2}$
$optionsResults_s = optionsResults_p$
**for** each $p \in OptionsResults$ **do**
  $UtLoss_p = U(lossInProfit_p)$
  $UtNbr_p = U(nbrJobs_p)$
  $Distance_p = \sqrt{(UtLoss_p - 1)^2 + (UtNbr_p - 1)^2}$
  **if** $Distance_p < minDistance$ **then**
    $minDistance = Distance_p$
    $optionsResults_s = optionsResults_p$
**return** $optionsResults_s$

# 4 The Renegotiation-based Rescheduling Procedure

Once a rescheduling option is selected, the *negoBusiness* triggers a renegotiation session with each *negoUser* representing the user whose job may be cancelled. We assume that the renegotiable issues are: (1) new deadline which is an extension beyond the first deadline and (2) compensation which is a discount by the business provider on the originally agreed-upon price, as a concession by the provider for avoiding the steep penalty envisaged by the original SLA violation. The values of the renegotiable issues (deadline, compensation) will be guided by the renegotiation decision-making strategy and will be based on the results of the selected rescheduling option. In this section, we present first the overall process for renegotiation. Then we present details about the strategies that will be followed by the *negoBusiness* and the *negoUser*.

## 4.1 The renegotiation overall process

A renegotiation session can be defined as the period covering the time when the interaction between negotiators begins until it stops. The renegotiation session terminates either with an agreement, and in this case the new SLA is applied, or without an agreement, in which case the initial SLA is applied. The different states of the renegotiation session, denoted *renegSessionState*, are: 1) *Active* (when the two parties are exchanging offers and counter-offers); 2) *Succeeded* (when the renegotiation session terminates with an agreement if one party accepts the offer received from his opponent); 3) *Failed* (when the renegotiation session terminates without an agreement). This last situation (*Failed*) occurs when one party rejects the opponent's offer or when the renegotiation deadline is reached.

The renegotiation-based rescheduling algorithm, Algorithm 4, takes as input the *potentialCancelJobs* list (included in the *optionsResults* returned by Algorithm 3). For each job that may be cancelled, the *negoBusiness* opens a renegotiation session with the *negoUser* that owns that job. The renegotiation sessions are triggered sequentially. The *negoBusiness* opens a new renegotiation session only if the current one is terminated (lines 3 and 4). If the renegotiation terminates with success, then the SLA is updated to include the new agreed upon deadline and the compensation (lines 5 and 6). If the renegotiation about the job $j$ fails then the *negoBusiness* must update the estimated completion time of the jobs that potentially are rescheduled after job $j$, in order to avoid the resource wastage due to unused time slots (lines 8 to 10). For that reason, the renegotiation is done sequentially, so that the *negoBusiness* can update the estimated completion time of the rescheduled jobs based on the renegotiation session's output.

**Algorithm 4** Pseudo-code for renegotiation-based rescheduling

**Require:** The list of potential cancelled jobs
**Ensure:** The results of each renegotiation session
1: **for** each $j \in potentialCancelJobs$ **do**
2: open renegotiation session $j$ with owner of job $j$
3: **while** $renegSessionState_j == Active$ **do**
4: **wait**
5: **if** $renegSessionState_j == Succeded$ **then**
6: update the $SLA_j$
7: **continue**
8: **else if** $renegSessionState_j == Failed$ **then**
9: **for** each $k \in$ rescheduled_jobs after j **do**
10: update $compT_k$ in $potentialCancelJobs_k$

## 4.2 The Decision-making Strategies for Renegotiation

During the renegotiation session, the *negoBusiness* and the *negoUser* automatically exchange offers and counter-offers according to their decision-making strategies. As explained in the Chapter III, the strategies are guided by the provisioning context information which be detailed in the following.
The renegotiation strategy should be designed to rapidly achieve agreement, since the participants are generally pressed when renegotiating after an SLA violation. For this reason, we assume that the new deadline proposed by the *negoBusiness* in the first round cannot be modified when exchanging offers and counter-offers. This is because the proposed deadline value is imposed by the rescheduling option selected. So the given deadline value is the best that the provider can offer to the client. In what follows, we present how the compensation value is evaluated and generated during the renegotiation session.

### 4.2.1 Decision-making by the *negoBusiness*

**Internal context** The $NegoBusiness$ uses the following information communicated by the business provider:

- The utility function value of a negotiable attribute $i$ with value $x$ can be calculated using equation IV.9 where the worst and best values are defined by the business provider before starting the negotiation as internal preference. In our scenario, based on the SLA model described in equation VI.1, the values $penaltyCost(deadline)$ and $fixedPenalty$ denote the best and worst values of compensation, respectively
- The *preferred* and *reserved* utility values as upper and lower bounds on the expected utility. The expected utility value consists of a tradeOff between minimizing the loss in profit and satisfying the client. The expected client utility can vary between 0 and 1. In the special case when the utility is equal to 1, the provider proposes a minimum compensation (the provider's best value) while still managing to relax the deadline. So, in this case the provider prefers minimizing the provider loss in profit over satisfying the client. And in the special case that the utility is equal to 0, the provider proposes to pay the fixed penalty as compensation while continuing to run the job. So, the provider doesn't minimize the provider loss, but instead satisfies the client by not cancelling his job. The preferred and reserved values are kept secret and are not know by the client.

**External context** As mentioned in Section 2, The *negoBusiness*'s decision-making relies on the external information form the monitor module, the Schedule manager and the VM provisioner. As mentioned in the Section 3, the monitor gives information about the resource(s)

affected. The *negoBusiness* gets information about the potential affected jobs from the Schedule manager. Based on those information, the agent chooses the best rescheduling option (minimizing the loss in profit and the number of cancelled jobs) based on the schedule manger and VM provisioner information. Finally, the *negoBusiness* could negotiate using the results of the rescheduling option selected.

**The offer evaluation strategy** The acceptance conditions of a received offer from the client during the renegotiation session are: 1) $U(compensation\_received) >= U(compensation\_proposed)$; and 2) $deadline\_received > compT$.
If the offer received from the client does not satisfy the two conditions mentioned above, the provider will propose a counter-offer using the *utility-based offer generation* strategy.

**The Strategy for Generation of Utility-based Offers** As mentioned earlier, the proposed new deadline will be equal to the estimated completion time included in $potentialCancelJobs$ list. Given the expected utility for the provider, the compensation utility value can be generated using equation IV.10. And the compensation value can be generated given the equation IV.9. In the first round, the provider agent generates the initial offer based on the provider's preferred utility. During the later rounds, the provider may back off from its preferred utility until reaching its reserved utility.

### 4.2.2 Decision-making by the Users

**Internal context** The $NegoUser$ uses the following information communicated by the user:

- The weights on the deadline and the compensation: Each attribute has a weight $w_i$ in the range $[0, 1]$ expressing its importance. For example, for high priority jobs, users may place more importance on the deadline than on compensation. In contrast, for low priority jobs, users may place more importance on the compensation than on the deadline.
- The range of each issue, delimited by its worst ($x_{worst}$) and best ($x_{best}$) values
- The preferred ($preferredUt$) and reserved ($reservedUt$) utility values, as bounds on the overall expected utility. Those clients having urgent business-critical jobs assign low value to the ($reservedUt$). This is because they prefer to accept the job along with a relaxed deadline and a smaller compensation, rather than having the job cancelled.
- The utility functions used to evaluate a received offer: the single attribute utility and the overall utility are calculated using the equations IV.9 and IV.10, respectively.

**The Strategy for Offer Evaluation** In our scenario, the client accepts an offer only if $U(offer_{received}) \geq reservedUt$. The client rejects an offer if $\exists$ *issue* $i, U(x_i) < 0$. Otherwise, the client proposes a counter-offer using the following strategy.

**The Strategy for Offer Generation** As mentioned earlier, the $negoUser$ does not change the deadline value proposed by the $negoBusiness$ when generating a counter-offer. Since the deadline utility is known (expressed by the $negoBusiness$'s initial offer), the compensation utility value can be generated from the expected overall utility using equation IV.10. As was the case for the $negoBusiness$, the $negoUser$ similarly starts by generating an offer according to the $preferredUt$ value, until reaching the $reservedUt$ value.

# 5 Evaluation

## 5.1 Experimental settings

We implemented the renegotiation framework (Figure VI.1) for SaaS provisioning using Multi-agent system. The details about the framework implementation are given in the evaluation Section of Chapter IV. We have extended the behavior of the agents in order to support proactive renegotiation. When there is unexpected event, the monitor agent sends an ACL message to the $negoBusiness$. After choosing the best rescheduling option (via the communication with the agents representing the Schedule manager and the VM provisioner), the $negoBusiness$ renegotiates with each $negoUser$ using the the bilateral version of the *the iterated contract net protocol*.

In the SaaS application provisioning process there are two phases: 1) Before the SLA establishment: the $negoBusiness$ tries to find a schedule satisfying the client's request, and decide whether to accept or reject the request. Once accepted, an SLA is signed between the two parties (between SaaS provider and client). 2) When an unexpected event occurs after the SLA establishment, as we have presented in Section 3, there are two steps. The first step happens when the monitor initially detects an unexpected event that may alter the initial schedule. The first step deals with choosing an option from several possible rescheduling options. In the second step, the $negoBusiness$ triggers a renegotiation session with each $NegoUser$ representing the user whose SLA may be violated.

Since we are interested in testing and validating the renegotiation approach, we assume in our experiments that:

- The first phase is done according to the negotiation-based scheduling algorithm described in chapter IV. Note that, any scheduling algorithm for SaaS provisioning can be used,

such as 1VMperAll, 1VMperJob, BinPacking heuristics, etc. However, we choose instead an SLA and a cost-aware scheduling algorithm in order to minimize the cost of the rented VMs by maximizing resource utilization. For each accepted job, the output of the first phase is an SLA with the required scheduling information. The scheduling information indicates where and when to put the job to satisfy the SLA.

- The first step of the second phase is not explicitly implemented. Instead, we generate an unexpected random event. We implement a rescheduling module simulator that generates the list of potential rescheduled jobs and their estimated completion time given an unexpected event. We assume that the jobs are rescheduled sequentially. The estimated completion for the job running can be generated randomly and for the other jobs using the following formula.

$$compT_j = compT_{jr} + \sum_{k \in \{\text{k between jr et j}\}} procT_{k,l} \tag{VI.2}$$

where $compT_{jr}$ denotes the completion time of the job running $jr$ at $t_{event}$. And $procT_{k,l}$ denotes the processing time of job $k$ on the VM of type $l$.
We assume that the rescheduling module simulator chooses the best rescheduling option.

## 5.2 Results and Analysis

Our objective is to evaluate the renegotiation-based application provisioning approach and to compare it to the basic scenario in which the provider cannot modify the established SLA. For the basic scenario, we assume that the provider tries to execute a rescheduling action (step 1) without any renegotiation. If the SLA is violated the job is cancelled and the SLA penalty is paid. We measure performance using two metrics: 1) the total loss in profit, expressing how much the provider loses when violating an already established SLAs; and 2) the number of cancelled jobs, the number of jobs whose completion time is beyond the agreed upon deadline for the original SLA.

We conduct three types of experiments in which we calculate the loss in profit and the number of cancelled jobs. For these experiments, we assume that each agent ($negoBusiness$ or $negoUser$) is able to generate only one offer during the renegotiation session, since the renegotiation must be done in a timely manner. Furthermore, we assume for the expected utility that the agent's reserved utility is equal to the preferred one. So the agents generate one offer according to their expected utility. If the opponent accepts the offer, the renegotiation ends with an agreement. Otherwise the renegotiation fails. This configuration (where preferred utility is equal to the reserved one) is the worst possible configuration in negotiation, since it is the least flexible. By choosing this configuration, we will be sure that for other configurations, our renegotiation

algorithm will perform better. Hence, when relaxing the expected utility, there is a greater chance of a request/offer being accepted, and so the number of successful renegotiation sessions will be increased.

For the first and the second experiments, we vary the expected utility for the business provider and the client, respectively, while injecting exactly one unexpected event (affecting only one VM). For the third experiment, we vary the number of resources affected by the unexpected event. Note that an event may lead to altering the initial scheduling of more than one VM. For example, a failure may affect many VMs.

### 5.2.1 Impact when varying the expected utility of the business provider

Figure VI.2 shows the different values obtained for the loss in profit and the number of cancelled jobs with respect to the provider expected utility. For those experiments, we generate clients and their initial request with expected utility equal to 0.1 (clients with business-critical jobs).

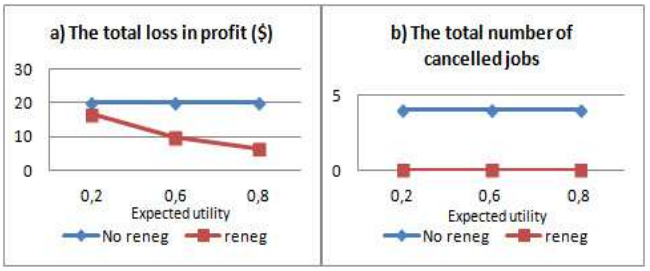


FIGURE VI.2: Impact of provider's expected utility variation

We observe that the loss in profit and the number of cancelled jobs using renegotiation is minimized compared to the basic scenario. Without renegotiation, the loss in profit and the number of cancelled jobs are constant regardless of the value of the provider utility. This is expected, since the *negoBusiness*'s strategy for handling unexpected events does not consider the value of the provider utility.

In Figure VI.2(a), the loss in profit (red curve) is decreasing when the business provider's expected utility increase. This is because the utility is related to the compensation paid to the client. The higher the utility, the less is the compensation that is paid, and so the loss in profit is also less. In Figure VI.2(b), the number of cancelled jobs (red curve) is constant regardless of the value of the provider utility, this is because the client's reserved utility is at the lower limit. This implies that the client will accept any offer from the business provider, even if the compensation is not at the upper limit (not at the upper bound for the provider utility). For those clients, a lower utility is nevertheless better than cancelling the job.

In the next experiments, we will vary the clients' expected utility.

### 5.2.2 Impact when varying the expected utility of the clients

Figure VI.3 shows the different values obtained for the loss in profit and the number of cancelled jobs, with respect to the clients' expected utility. For those experiments, the provider's expected utility is equal to 0.6. We note, as in Figure VI.2, that the loss in profit and the number of cancelled jobs are constant in the basic scenario, since the basic scenario does not take into account client satisfaction.

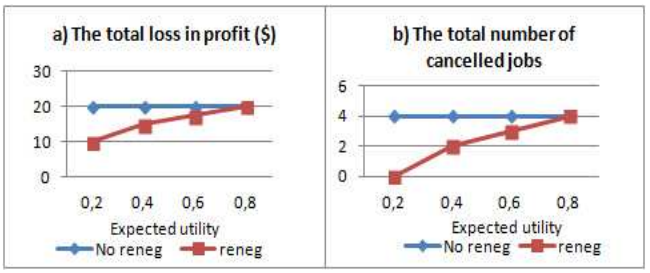


FIGURE VI.3: Impact of variation of clients' expected utility

With renegotiation, we notice that the loss in profit and the number of cancelled jobs increase when the client expected utility increases. For the users with low utility values, the renegotiation approach performs much better than the basic one. But, for users with high utility values, the renegotiation approach results are the same as the basic one. So, when increasing the clients' expected utility, the renegotiation approach performance tends to the performance of the basic one. In contrast, when the expected utility is low, the client has a high-priority business-critical job, and so the *negoUser* accepts any renegotiation offer in order to assure the continuity of the user's business. In contrast, the *negoUser* representing the client with a high expected utility (i.e., having a less business-critical job) may choose to not accept a renegotiation offer. In this case, the client prefers that the provider should pay the penalty and cancel the job.

### 5.2.3 Impact as the number of resources are varied

Figure VI.4 shows the different values obtained for the loss in profit and the number of cancelled jobs with respect to the number of affected resources. For those experiments, the provider and the client expected utility are equal to 0.6 and 0.1, respectively.

We notice that the loss in profit (with and without renegotiation) and the number of cancelled jobs (without renegotiation) increase when the number of affected resources increases. Further, when the unexpected event affects many VMs, the number of rescheduled jobs increases which lead to a potentially increased number of cancelled jobs. Consequently, the total loss in profit will increase. In Figure VI.4(b), the number of cancelled jobs is equal to zero, regardless of the number of resources. This is because, in our configuration, we generate clients whose jobs are highly business-critical. So the clients always accept the renegotiation requests.

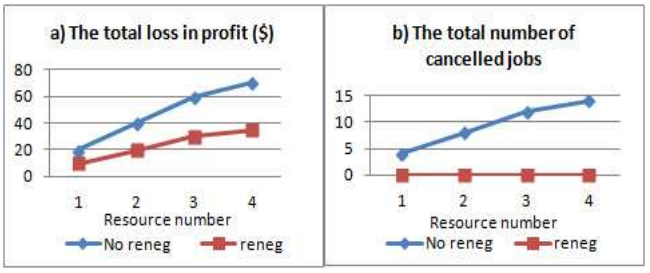


FIGURE VI.4: Impact of variation of number of resources

For the three experiments, we conclude that: 1) our approach performs better than the basic one in terms of profit and the number of cancelled jobs when the clients' jobs are highly business-critical (low expected client utility); and 2) the performance of our approach tends toward the basic one when the clients have jobs that are less business-critical (when the clients' expected utility is high). Thus in the second case, the clients do not accept a renegotiation, and prefer to enforce the initial SLA.

# 6 Conclusion

In this chapter, we have described an SLA renegotiation-based approach to proactively handle SLA violations. The renegotiation process is based on the interaction between *negoBusiness* agent (representing the business provider) and business provider's provisioning modules (Schedule manager, VM provisioner, Monitor). When detecting an unexpected event by the monitor, the business provider may take rescheduling actions to avoid SLA violation. We proposed an algorithm to select the best rescheduling option aiming to minimize the loss in profit and the number of cancelled jobs. The decision-making strategies are based on a utility function for the business provider and scheduling information generated by the rescheduling option chosen before renegotiation. The resulting decision-making model makes possible a win-win situation (ensuring continuity of service and minimizing SLA penalties costs). The experimental evaluation demonstrates the benefit of an SLA renegotiation approach as compared with enforcing the original SLA.

# Conclusions and Future Work

## 1 Contributions and Research Summary

The cloud market is nowadays a complex environment where business providers need to maximize their monetary profit and end-users look for the most efficient services with the lowest prices. For efficient cloud provisioning, both providers and users should be satisfied in spite of their conflicting needs. Negotiation is an appealing solution to solve conflicts and enable reaching satisfactory agreements. However, to be efficient, the negotiation should consider the cloud provisioning properties (e.g., two interaction levels) and complexities related to dynamicity (e.g., resource availability variation, dynamic pricing) which impact highly the success of the negotiation.

Automated negotiation for an efficient cloud provisioning represents a major challenge that is not adequately treated despite the several existing researches. In order to address these challenges, this thesis provides several novel contributions:

- *The negotiation-based framework applied to cloud provisioning*

  In order to build a negotiation framework suitable for cloud provisioning, we have first proposed a generic negotiation-based provisioning process showing the interaction between the negotiation and other provisioning activities. Then, we propose a generic negotiator model able to act on behalf of any cloud actor (end-user, business provider, resource provider) and can be used for any negotiation scenario. That model was instantiated among cloud layers resulting in a multi-layer and dynamic negotiation framework for cloud provisioning. What makes our framework differs from others is the fact that it considers cloud provisioning properties (multi-layer, dynamic provisioning context).

- *Bilateral negotiation for an efficient SaaS application provisioning*

  For an efficient application provisioning that satisfies end-users and that optimizes business provider's profit, we have enhanced an existing provisioning approach with negotiation capabilities. The negotiation is carried out between end-users and a business provider. When

it is not possible to schedule the user's request, the business provider tries to find an alternative schedule satisfying both parties. The alternative schedules are generated through the communication with other provisioning modules such as the admission control, scheduler and VM provisioner. We have proposed two decision-making strategies that can be followed according to the business provider's goal (i.e., preferences).
In contrast to existing SaaS negotiation approaches, our approach is based on the communication with the other provisioning modules in order to optimize business provider's profit and maximize customer satisfaction. Furthermore, our approach differs from existing provisioning approaches by the fact that it considers SLA negotiation between end-users and the business provider in order to bring more flexibility to the provisioning process.

- *Adaptive and concurrent negotiation for efficient SaaS application provisioning*
  Due to the gap between the users' requirements and the business provider's capabilities at negotiation time, the negotiation may not reach a compromise. So, in order to maximize the number of clients, we have proposed an adaptive and concurrent negotiation approach as an extension to the bilateral one previously described. We have proposed to harness the workload changes in terms of resource availability and pricing in order to renegotiate simultaneously with non-accepted users. The renegotiation is launched when there is a change in the elements considered in the first negotiation. The proposed renegotiation decision-making strategy aims to maximize resource utilization and also customer satisfaction.
  Our proposed approach differs from existing ones by the fact that it considers renegotiation before SLA establishment according to workload changes in terms of resource availability and pricing.

- *Proactive renegotiation for an efficient SaaS provisioning*
  In order to handle potential SLA violation, we have proposed a proactive renegotiation approach after SLA establishment. When detecting an unexpected event by the monitor, the business provider may take rescheduling actions to avoid SLA violation. We proposed an algorithm to select the best rescheduling option aiming to minimize the loss in profit and the number of cancelled jobs. The decision-making strategies are based on scheduling information generated by the rescheduling option chosen before renegotiation. The resulting decision-making model makes possible a win-win situation (ensuring continuity of service and minimizing SLA penalties costs). Our proposed approach differs from existing ones by the fact that it proposes a concrete renegotiation decision-making model for SaaS provisioning aiming to minimize the loss in profit and ensure the continuity of the service.

# 2 Future Directions

Besides the aforementioned contributions, several perspectives are still to be investigated in order to improve our work. In what follows, we present short and long-term perspectives that extend and enhance our work.

## 2.1 Short-term Perspectives

In order to improve the proposed approach, the following short-term perspectives can be investigated:

- Implementing negotiation capabilities at resource level (*level 1*)
  In this thesis, we have concentrated on business level while considering dynamic and on-demand resource provisioning from different resources providers. So, we have implemented and tested only the negotiation between end-users and business provider (*level 2*). As a first step towards enhancing the proposed work, we plan to implement negotiation between business providers and resource providers. To do so, existing resource negotiation strategies can be reused (e.g., [21], [19], [92]). We are interested in studying the dependencies between the negotiation strategies of the different levels (*level 1* and *level 2*) while considering the different types of coordination used by the business provider (presented in Chapter III).
- Designing negotiation strategies for end-users considering different business providers
  In this thesis, we have focused on the business provider's negotiation strategies while we have reused existent negotiation strategies for end-users (e.g, tradeoff strategies [22]). The second short-term perspective consists in designing negotiation strategies for end-users considering different business providers. In fact, the negotiation may be associated to selection approaches in order to assist end-users to find the best business providers satisfying their needs [42].
- Implementing the rescheduling step when detecting an unexpected event (before renegotiation)
  In this thesis, the rescheduling step presented in Chapter VI was not implemented. We plan to further investigate the rescheduling options when detecting an expected event. Then, we aim to study the impact when the rescheduling option is varied in an effective large scale environment.
- Considering real-data
  In this thesis, all experiments were conducted on simulation studies. We plan to test our

approach in a real cloud environment. Furthermore, we are interested in studying and testing others negotiation scenarios with new requirements and others negotiation issues.

- Developing a module for assessing negotiation time
  Finally, we plan to develop a module for assessing the negotiation time. This could be done by calculating the time taken by negotiation for different scenarios. In fact, before engaging in a negotiation, it would be better to know whether it is worth it to negotiate or not (for instance, the negotiation may be useless if the negotiation time is high and the negotiated service is deadline-sensitive).

## 2.2 Long-term Perspectives

The following perspectives aim to extend our proposed negotiation approaches:

- Developing the self-negotiation aspect
  Given the existing and the implemented strategies, we aim to design a tool to assist negotiator to choose the best strategy (combination of tactics, parameters considered by the strategy) according to the negotiation scenario. To do so, we plan to use the *autonomic computing* principles [97–99]. The *Knowledge database* may contain the negotiation strategies, the internal context elements and the already established SLAs. The *Monitor* should be able to communicate with external context elements such as the market prospector and updates the knowledge base according to the information collected. The *Analyze* and *Plan* modules should be able to analyze the negotiation scenario and choose the best negotiation strategy. Finally, the *Execute* module should be able to evaluate/generate offer(s) according to the chosen strategy.

- Implementing the negotiator tool generator
  We are interested in designing and implementing a negotiator tool generator. The negotiator generator tool should be able to generate automatically a negotiator that is able to negotiate autonomously regardless of the negotiation scenario. The negotiator generator may take as input the application domain, the negotiated service and the negotiator's preferences. The core of the negotiator tool will be based on the model presented in Chapter III.

# Appendix A

# Example of negotiation protocols

## 1 Alternate offers protocol

Figure A.1 shows the allowed interactions between two participants (*proposer* and *responder*) in a negotiation session following the alternate offers protocol.
The negotiation process is started by the *proposer* who sends an *initiate* message to the *responder*. The *responder* replies with the negotiation identifier (*negotiationID*).Then, the *proposer* sends his/her proposal (*submitProposal*). The *responder* evaluates the received proposal and replies by accepting, rejecting or sending a counter-offer. In case the response is a counter-offer, the *proposer* has the same options. The iteration is repeated until one party accepts the received proposal or aborts the negotiation session by sending a *reject* message. After receiving an *accept* message, the other party has to send a *confirm* message and receives a *confirm-acceptance* message as a response.

## 2 Iterated Contract Net Protocol

Figure A.2 shows the protocol flows between one *initiator* and $n$ participants in a negotiation session following the iterated contract net protocol.
The *initiator* starts the negotiation process by sending $m$ initial call for proposals (cfp). The participant can refuse or propose a message. So of the $n$ participants, the *initiator* can receive $k$ propose messages and $j$ refuse messages. Of the $k$ proposals, if there are acceptable messages (final iteration), the *initiator* may accept $p$ of the bids and reject others. Alternatively the initiator may decide to iterate the process by sending a revised cfp to $l$ participants and so rejecting $k - l$ participants. Then, if the action is achieved, the participant sends an *inform*

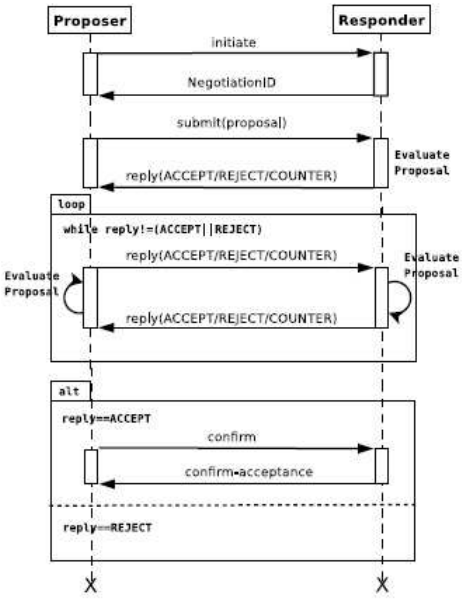


FIGURE A.1: Alternate offers protocol [3]

message. Else, the participant sends a *failure* message.

The negotiation process terminates if (i) the *initiator* rejects all the received proposals, OR (ii) the initiator accepts at least one proposal, OR (iii) all the participants refuse the bid.

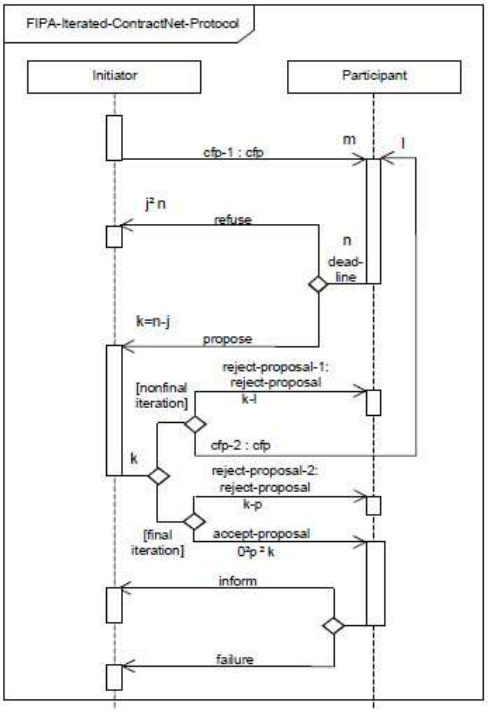


FIGURE A.2: Iterated Contract Net Protocol [4]

# Appendix B

# Multi-Agent System

## 1 Definitions

### Agent Definition

Several definitions had been proposed, however most researchers agreed on the following definition proposed by Wooldridge et al. [100]:
*"An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives."* According to that definition, an agent can have the following properties [101]:

- Autonomy: an agent operates without human interaction and has control over its actions.
- Social ability: an agent communicates with other agents using an *agent-communication language*.
- Reactivity: an agent perceives the environment and acts according to environment's changes.
- pro-activeness: an agent is able to take actions according to its goal.

### MAS definition

In [102], A MAS system is defined as follows:
*"A multi-agent system is a loosely coupled network of problem-solving entities (agents) that work together to find answers to problems that are beyond the individual capabilities or knowledge of*

*each entity (agent).”*

Ferber defines the following elements composing a MAS [103]:

- An environment **E** representing a space having generally a metric.
- A set of objects **O**. Each object can have a position in E at a given moment. Those objects are passive, in other words the agents can perceive, create, modify and destroy them.
- A set of agent **A**, agents are special objects representing the active entities of the system.
- A set of relation **R**, which unites objects between them.
- A set of operators **Op** allowing the agents of A to perceive, produce, consume, transform and manipulate objects of O.
- Operators called the laws of the universe. Those operators represents the application of these operations and the reaction of the world to the modification attempt.

To summarize, A MAS consists of a set of agents, interacting with each others via messages, in a common environment where the agents can act and cooperate to achieve their objectives.

## 2 JADE platform

Many platforms have been developed to implement agent-based systems (e.g, MADKIT [1], MASON [2]). Based on MAS platforms comparison [104–106], we have chosen JADE plateform. JADE is compliant to FIPA specification which facilitate interoperability among different MAS. Also, Jade platform provides many graphical tools for development and debugging [107].

### JADE architecture

Figure B.1 illustrates the architectural elements of the JADE platform. The JADE platform is composed of containers which can be launched on different hosts. Each container can contain zero or more agents. For instance, as shown in figure B.1, container ”Container 1” in host ”Host 3” contains agents ”A2” and ”A3”. The platform contains a special container called ”Main Container”. The main container differs from the other containers as:

- It is the first container to start in the platform and the other containers register to it at bootstrap time.

[1] www.madkit.net
[2] https://cs.gmu.edu/ eclab/projects/mason/

- It includes two special agents. The first one called Agent Management System (AMS) which responsible of platform management such as starting and killing agent or shutting down the platform. Other agents request such actions to the AMS. The second special agent is called Directory Facilitator (DF) and provides Yellow Pages service where the service offered by the different agents can be published.

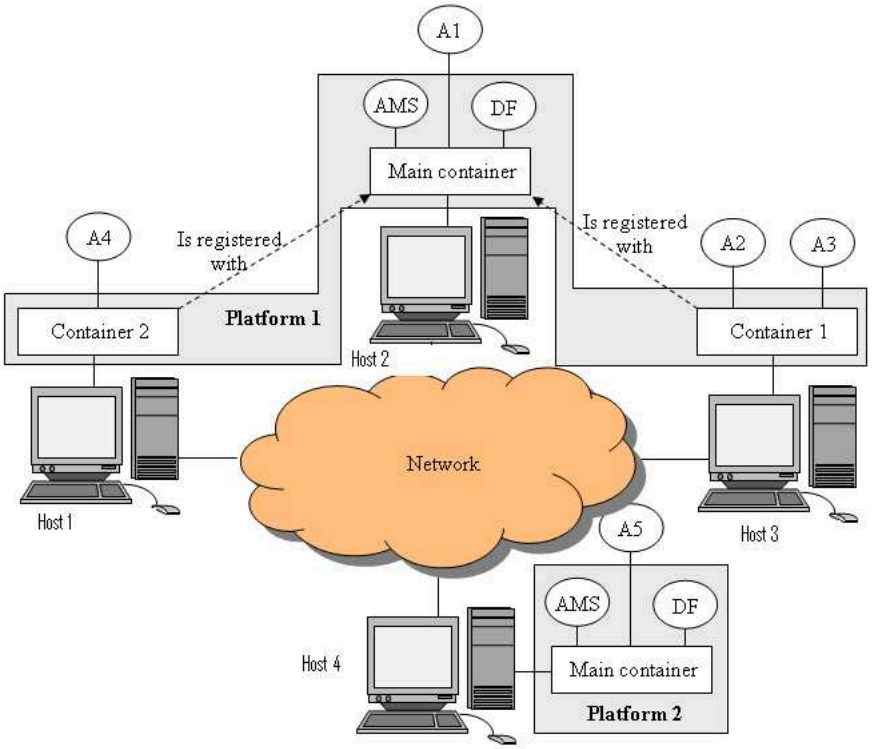


FIGURE B.1: The JADE Architecture [5]

## 3 Communication between agents

Agents can communicate transparently regardless of their location (same container, different containers, same platform, different platforms). The communication between agents is based on asynchronous message passing. The message format is defined according to ACL-FIPA specifications. An ACL Message contains the following attributes:

- the sender
- the receiver(s)
- the communicative act (also called performative) represents the intention of the sender of the message. FIPA defined 22 communicative acts such as inform, request, agree, propose,

query. Each performative has its own semantics. In our thesis, we use the performatives related to the FIPA iterated contract net protocol (e.g, cfp, propose, inform, refuse).

- the content contains the information that the sender would like to send to the receiver. The message content depends on the type of the communicative act. In our work the content may contain the QoS attributes and their values (as part of the SLA).

# Bibliography


[1] Junliang Chen, Chen Wang, Bing Bing Zhou, Lei Sun, Young Choon Lee, and Albert Y. Zomaya. Tradeoffs between profit and customer satisfaction for service provisioning in the cloud. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, HPDC '11, pages 229–238, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0552-5. doi: 10.1145/1996130.1996161. URL http://doi.acm.org/10.1145/1996130.1996161.

[2] Kyriakos Kritikos, Barbara Pernici, Pierluigi Plebani, Cinzia Cappiello, Marco Comuzzi, Salima Benrernou, Ivona Brandic, Attila Kertész, Michael Parkin, and Manuel Carro. A survey on service quality description. *ACM Comput. Surv.*, 46(1):1:1–1:58, July 2013. ISSN 0360-0300. doi: 10.1145/2522968.2522969. URL http://doi.acm.org/10.1145/2522968.2522969.

[3] Srikumar Venugopal, Xingchen Chu, and Rajkumar Buyya. A negotiation mechanism for advance resource reservations using the alternate offers protocol. In Hans van den Berg and Gunnar Karlsson, editors, *IWQoS*, pages 40–49. IEEE, 2008. ISBN 978-1-4244-2084-1. URL http://dblp.uni-trier.de/db/conf/iwqos/iwqos2008.html#VenugopalCB08.

[4] FIPA Interaction Protocols, http://www.fipa.org/repository/ips.php3.

[5] Jade Site: Java Agent DEvelopment Framework, http://jade.tilab.com/.

[6] A. Bestavros and O. Krieger. Toward an open cloud marketplace: Vision and first steps. *IEEE Internet Computing*, 18(1):72–77, Jan 2014. ISSN 1089-7801. doi: 10.1109/MIC.2014.17.

[7] Rodrigo N. Calheiros, Rajiv Ranjan, and Rajkumar Buyya. Virtual machine provisioning based on analytical performance and QoS in cloud computing environments. In *Proceedings of the 2011 International Conference on Parallel Processing*, ICPP '11, pages 295–304, Washington, DC, USA, 2011. IEEE Computer Society. ISBN 978-0-7695-4510-3. doi: 10.1109/ICPP.2011.17. URL http://dx.doi.org/10.1109/ICPP.2011.17.

[8] Rajiv Ranjan, Liang Zhao, Xiaomin Wu, and Anna Liu. *Cloud Computing Principles, Systems and Applications*, chapter Peer-to-Peer Cloud Provisioning: Service Discovery and Load-Balancing, pages pp.195–217. Springer, 2010.

[9] Linlin Wu and Rajkumar Buyya. Service level agreement (sla) in utility computing systems. *CoRR*, 14:286–310, 2010.

[10] Edwin Yaqub, Ramin Yahyapour, Philipp Wieder, Constantinos Kotsokalis, Kuan Lu, and Ali Imran Jehangiri. Optimal negotiation of service level agreements for cloud-based services through autonomous agents. In *Proceedings of the 2014 IEEE International Conference on Services Computing*, SCC '14, pages 59–66, Washington, DC, USA, 2014. IEEE Computer Society. ISBN 978-1-4799-5066-9. doi: 10.1109/SCC.2014.17. URL http://dx.doi.org/10.1109/SCC.2014.17.

[11] N. R. Jennings, P. Faratin, A. R. Lomuscio, S. Parsons, M. Wooldridge, and C. Sierra. Automated negotiation: prospects, methods and challenges. *Intern. J. of Group Decision and Negotiation*, 10(2):199–215, 2001.

[12] P. Church, A. Wong, M. Brock, and A. Goscinski. Toward exposing and accessing hpc applications in a saas cloud. In *2012 IEEE 19th International Conference on Web Services*, pages 692–699, June 2012. doi: 10.1109/ICWS.2012.119.

[13] A. Omezzine, S. Tazi, N. Bellamine, B. Saoud, K. Drira, and G. Cooperman. Towards a dynamic multi-level negotiation framework in cloud computing. In *Cloud Technologies and Applications (CloudTech), 2015 International Conference on*, pages 1–8, June 2015. doi: 10.1109/CloudTech.2015.7336999.

[14] Linlin Wu, Saurabh Kumar Garg, Rajkumar Buyya, Chao Chen, and Steve 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. IEEE, 2013.

[15] Seokho Son, Dong-Jae Kang, Seyoung Phillip Huh, Won-Young Kim, and Wan Choi. Adaptive trade-off strategy for bargaining-based multi-objective SLA establishment under varying cloud workload. *The Journal of Supercomputing*, 72(4):1597–1622, 2016. doi: 10.1007/s11227-016-1686-y. URL http://dx.doi.org/10.1007/s11227-016-1686-y.

[16] Melanie Siebenhaar, Ulrich Lampe, Dieter Schuller, Ralf Steinmetz, et al. Concurrent negotiations in cloud-based systems. In *Economics of Grids, Clouds, Systems, and Services*, pages 17–31. Springer, 2012.

[17] Guoming Lai and Katia Sycara. A generic framework for automated multi-attribute negotiation. *Group Decision and Negotiation*, 18(2):169, Jul 2008. ISSN 1572-9907. doi: 10.1007/s10726-008-9119-9. URL http://dx.doi.org/10.1007/s10726-008-9119-9.

[18] Peyman Faratin, Carles Sierra, and Nicholas R. Jennings. Negotiation decision functions for autonomous agents. *Robotics and Autonomous Systems*, 24(3-4):159–182, 1998. doi: 10.1016/S0921-8890(98)00029-3. URL http://dx.doi.org/10.1016/S0921-8890(98)00029-3.

[19] Amir Vahid Dastjerdi and Rajkumar Buyya. An autonomous reliability-aware negotiation strategy for cloud computing environments. In *CCGRID*, pages 284–291. IEEE Computer Society, 2012. ISBN 978-1-4673-1395-7. URL http://dblp.uni-trier.de/db/conf/ccgrid/ccgrid2012.html#DastjerdiB12.

[20] Linlin Wu, Saurabh Kumar Garg, and Rajkumar Buyya. SLA-based admission control for a software-as-a-service provider in cloud computing environments. *J. Comput. Syst. Sci.*, 78(5):1280–1299, September 2012. ISSN 0022-0000. doi: 10.1016/j.jcss.2011.12.014. URL http://dx.doi.org/10.1016/j.jcss.2011.12.014.

[21] Seokho Son and Kwang Mong Sim. A price- and-time-slot-negotiation mechanism for cloud service reservations. *IEEE Trans. Systems, Man, and Cybernetics, Part B*, 42(3):713–728, 2012. doi: 10.1109/TSMCB.2011.2174355. URL http://dx.doi.org/10.1109/TSMCB.2011.2174355.

[22] X. Zheng, P. Martin, and K. Brohman. Cloud service negotiation: Concession vs. tradeoff approaches. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pages 515–522, May 2012. doi: 10.1109/CCGrid.2012.94.

[23] Zhipiao Liu, Shangguang Wang, Qibo Sun, Hua Zou, and Fangchun Yang. Cost-aware cloud service request scheduling for saas providers. *The Computer Journal*, page bxt009, 2013.

[24] Philipp Leitner, Waldemar Hummer, Benjamin Satzger, Christian Inzinger, and Schahram Dustdar. Cost-efficient and application SLA-aware client side request scheduling in an infrastructure-as-a-service cloud. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 213–220. IEEE, 2012.

[25] L. Wu, S. K. Garg, and R. Buyya. SLA-based resource allocation for software as a service provider (saas) in cloud computing environments. In *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 195–204, May 2011. doi: 10.1109/CCGrid.2011.51.

[26] Linlin Wu, Saurabh Kumar Garg, Steve Versteeg, and Rajkumar Buyya. SLA-based resource provisioning for hosted software-as-a-service applications in cloud computing environments. *IEEE Transactions on services computing*, 7(3):465–485, 2014.

[27] Ahmad Fadzil M Hani, Irving Vitra Paputungan, and Mohd Fadzil Hassan. Renegotiation in service level agreement management for a cloud-based system. *ACM Computing Surveys (CSUR)*, 47(3):51, 2015.

[28] D. Ouelhadj, J. Garibaldi, J. Maclaren, R. Sakellariou, and K. Krishnakumar. A multi-agent infrastructure and a service level agreement negotiation protocol for robust scheduling in grid computing. In *In Advances in Grid Computing - EGC 2005, volume 3470 of Lecture Notes in Computer Science*, pages 651–660. Springer Verlag, 2005.

[29] Sanaa Sharaf and Karim Djemame. Extending ws-agreement to support renegotiation of dynamic grid slas. In *eChallenges e-2010 Conference*, pages 1–8. IEEE, 2010.

[30] Michael Parkin, Peer Hasselmeyer, and Bastian Koller. An sla re-negotiation protocol. In *Proceedings of the 2nd Non Functional Properties and Service Level Agreements in Service Oriented Computing Workshop (NFPSLA-SOC08), CEUR Workshop Proceedings, ISSN 1613-0073, Volume 411*. Citeseer, 2008.

[31] Giuseppe Di Modica, Orazio Tomarchio, and Lorenzo Vita. Dynamic slas management in service oriented environments. *Journal of Systems and Software*, 82(5):759–771, 2009.

[32] Ahmad Fadzil M Hani, Irving Vitra Paputungan, and M Fadzil Hassan. Service level agreement renegotiation framework for trusted cloud-based system. In *Future Information Technology*, pages 55–61. Springer, 2014.

[33] Werner Mach and Erich Schikuta. A generic negotiation and re-negotiation framework for consumer-provider contracting of web services. In *Proceedings of the 14th International Conference on Information Integration and Web-based Applications & Services*, pages 348–351. ACM, 2012.

[34] Michael Hogan, Fang Liu, Annie Sokol, and Jin Tong. Nist cloud computing standards roadmap. Technical report, National Institute of Standards and Technology, 2011.

[35] Borko Furht. *Handbook of Cloud Computing*, chapter Cloud Computing Fundamentals, page pp 319. Springer, 2010.

[36] Amit Nathani, Sanjay Chaudhary, and Gaurav Somani. Policy based resource allocation in iaas cloud. *Future Gener. Comput. Syst.*, 28(1):94–103, January 2012. ISSN 0167-739X. doi: 10.1016/j.future.2011.05.016. URL http://dx.doi.org/10.1016/j.future.2011.05.016.

[37] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The eucalyptus open-source cloud-computing system. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '09, pages 124–131, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3622-4. doi: 10.1109/CCGRID.2009.93. URL http://dx.doi.org/10.1109/CCGRID.2009.93.

[38] Richard Lawley, Keith Decker, Michael Luck, Terry Payne, and Luc Moreau. Automated negotiation for grid notification services. In *Euro-Par 2003 Parallel Processing*, 2003.

[39] Ariel Rubinstein. Perfect equilibrium in a bargaining model. *Econometrica*, 50(1):97–109, 1982. ISSN 00129682, 14680262. URL http://www.jstor.org/stable/1912531.

[40] Radu Prodan, Marek Wieczorek, and Hamid Mohammadi Fard. Double auction-based scheduling of scientific applications in distributed grid and cloud environments. *J. Grid Comput.*, 9(4):531–548, 2011. doi: 10.1007/s10723-011-9196-x. URL http://dx.doi.org/10.1007/s10723-011-9196-x.

[41] Kwang Mong Sim. *Towards Complex Negotiation for Cloud Economy*, pages 395–406. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-13067-0. doi: 10.1007/978-3-642-13067-0_42. URL http://dx.doi.org/10.1007/978-3-642-13067-0_42.

[42] Mahboobeh Moghaddam and Joseph G. Davis. Service selection in web service composition: A comparative review of existing approaches. In Athman Bouguettaya, Quan Z. Sheng, and Florian Daniel, editors, *Web Services Foundations*, pages 321–346. Springer, 2014. ISBN 978-1-4614-7518-7. URL http://dblp.uni-trier.de/db/books/collections/wsf2014.html#MoghaddamD14.

[43] Shaheen S. Fatima, Michael Wooldridge, and Nicholas R. Jennings. Multi-issue negotiation under time constraints. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1*, AAMAS '02, pages 143–150, New York, NY, USA, 2002. ACM. ISBN 1-58113-480-0. doi: 10.1145/544741.544775. URL http://doi.acm.org/10.1145/544741.544775.

[44] Mohammed Alhamad, Tharam Dillon, and Elizabeth Chang. Conceptual sla framework for cloud computing. In *IEEE international conference on digital ecosystems and technologies (DEST 2010)*, 2010.

[45] Jun Yan, Ryszard Kowalczyk, Jian Lin, Mohan B. Chhetri, Suk Keong Goh, and Jianying Zhang. Autonomous service level agreement negotiation for service composition provision. *Future Gener. Comput. Syst.*, 23(6):748–759, July 2007. ISSN 0167-739X. doi: 10.1016/j.future.2007.02.004. URL http://dx.doi.org/10.1016/j.future.2007.02.004.

[46] M. Macas and J. Guitart. Using resource-level information into nonadditive negotiation models for cloud market environments. In *2010 IEEE Network Operations and Management Symposium - NOMS 2010*, pages 325–332, April 2010. doi: 10.1109/NOMS.2010.5488485.

[47] Amir Vahid Dastjerdi. *QoS-aware and Semantic-based Service Coordination for Multi-Cloud Environments*. PhD thesis, THE UNIVERSITY OF MELBOURNE, 2013.

[48] Kwang mong Sim. Towards a unifying multilateral cloud negotiation strategy. In *International MultiConference of Engineers and Computer Scientists*, 2013.

[49] Farhana H. Zulkernine, Patrick Martin, Chris Craddock, and Kirk Wilson. A policy-based middleware for web services SLA negotiation. In *ICWS*, pages 1043–1050. IEEE Computer Society, 2009.

[50] Farhana H. Zulkernine and Patrick Martin. An adaptive and intelligent SLA negotiation system for web services. *IEEE T. Services Computing*, 4(1):31–43, 2011. URL http://dblp.uni-trier.de/db/journals/tsc/tsc4.html#ZulkernineM11.

[51] Serban Radu. *An Adaptive Negotiation Multi-Agent System for e-Commerce Applications*. PhD thesis, University Politehnica of Bucharest, 2013.

[52] Ioan Salomie Tudor Cioara Ionut Anghel Georgiana Copil, Daniel Moldovan and Diana Borza. Cloud sla negotiation for energy saving  a particle swarm optimization approach. In *Intelligent Computer Communication and Processing (ICCP)*, 2012.

[53] Jamal Bentahar, Zakaria Maamar, Wei Wan, Djamal Benslimane, Philippe Thiran, and Sattanathan Subramanian. Agent-based communities of web services: an argumentation-driven approach. *Service Oriented Computing and Applications*, 2(4):219–238, 2008. URL http://dblp.uni-trier.de/db/journals/soca/soca2.html#BentaharMWBTS08.

[54] Francesca Toni, Mary Grammatikou, Stella Kafetzoglou, Leonidas Lymberopoulos, Symeon Papavassileiou, Dorian Gaertner, Maxime Morge, Stefano Bromuri, Jarred Mcginnis, Kostas Stathis, Vasa Curcin, Moustafa Ghanem, and Li Guo. The argugrid platform: An overview. In *Proceedings of the 5th international workshop on Grid Economics and Business Models*, GECON '08, pages 217–225, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-85484-5. doi: 10.1007/978-3-540-85485-2_18. URL http://dx.doi.org/10.1007/978-3-540-85485-2_18.

[55] Iyad Rahwan and Kate Larson. *Argumentation in Artificial Intelligence*, chapter Argumentation and Game Theory, pages 321–339. Springer Science+Business Media, 2009.

[56] Iyad Rahwan and Kate Larson. Mechanism design for abstract argumentation. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 2*, AAMAS '08, pages 1031–1038, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-0-9817381-1-6. URL http://dl.acm.org/citation.cfm?id=1402298.1402365.

[57] Nicolas Maudet, Simon Parsons, and Iyad Rahwan. *Argumentation in Multi-Agent Systems: Context and Recent Developments*, pages 1–16. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-75526-5. doi: 10.1007/978-3-540-75526-5_1. URL http://dx.doi.org/10.1007/978-3-540-75526-5_1.

[58] Iyad Rahwan, Sarvapali D. Ramchurn, Nicholas R. Jennings, Peter Mcburney, Simon Parsons, and Liz Sonenberg. Argumentation-based negotiation. *Knowl. Eng. Rev.*, 18(4): 343–375, December 2003. ISSN 0269-8889. doi: DOI:10.1017/S0269888904000098. URL http://dx.doi.org/DOI:10.1017/S0269888904000098.

[59] Stella Heras, Fernando de la Prieta, Sara Rodrguez, Javier Bajo, Vicente J. Botti, and Vicente Julin. The role of argumentation on the future internet: Reaching agreements on clouds. In Sascha Ossowski, Francesca Toni, and George A. Vouros, editors, *AT*, volume 918 of *CEUR Workshop Proceedings*, pages 393–407. CEUR-WS.org, 2012. URL http://dblp.uni-trier.de/db/conf/at/at2012.html#HerasPRBBJ12.

[60] Stella M. Heras Barbera. *CASE-BASED ARGUMENTATION IN AGENT SOCIETIES*. PhD thesis, UNIVERSITAT POLITECNICA DE VALENCIA, 2011.

[61] Adel Nadjaran Toosi, Rodrigo N. Calheiros, and Rajkumar Buyya. Interconnected cloud computing environments: Challenges, taxonomy, and survey. *ACM Comput. Surv.*, 47(1): 7:1–7:47, May 2014. ISSN 0360-0300. doi: 10.1145/2593512. URL http://doi.acm.org/10.1145/2593512.

[62] Salvatore Venticinque, Rocco Aversa, Beniamino Di Martino, Massimilano Rak, and Dana Petcu. *A Cloud Agency for SLA Negotiation and Management*, pages 587–594. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-21878-1. doi: 10.1007/978-3-642-21878-1_72. URL http://dx.doi.org/10.1007/978-3-642-21878-1_72.

[63] A. AuYoung, L. Grit, J. Wiener, and J. Wilkes. Service contracts and aggregate utility functions. In *2006 15th IEEE International Conference on High Performance Distributed Computing*, pages 119–131, 2006. doi: 10.1109/HPDC.2006.1652143.

[64] Stéphane Genaud and Julien Gossa. Cost-wait trade-offs in client-side resource provisioning with elastic clouds. In *IEEE International Conference on Cloud Computing, CLOUD 2011, Washington, DC, USA, 4-9 July, 2011*, pages 1–8, 2011. doi: 10.1109/CLOUD.2011.23. URL http://dx.doi.org/10.1109/CLOUD.2011.23.

[65] G. Le, K. Xu, and J. Song. Dynamic resource provisioning and scheduling with deadline constraint in elastic cloud. In *2013 International Conference on Service Sciences (ICSS)*, pages 113–117, April 2013. doi: 10.1109/ICSS.2013.18.

[66] Radu Prodan and Marek Wieczorek. Negotiation-based scheduling of scientific grid workflows through advance reservations. *J. Grid Comput.*, 8(4):493–510, 2010. doi: 10.1007/s10723-010-9165-9. URL http://dx.doi.org/10.1007/s10723-010-9165-9.

[67] Nooruldeen Nasih Qader Bahador Shojaiemehr, Amir Masoud Rahmani. Cloud computing service negotiation: A systematic review. *Computer Standards & Interfaces*, 2017.

[68] Kwang Mong Sim. Grid resource negotiation: survey and new directions. *Trans. Sys. Man Cyber Part C*, 40(3):245–257, May 2010. ISSN 1094-6977. doi: 10.1109/TSMCC.2009.2037134. URL http://dx.doi.org/10.1109/TSMCC.2009.2037134.

[69] Alexander Keller and Heiko Ludwig. The wsla framework: Specifying and monitoring service level agreements for web services. *J. Netw. Syst. Manage.*, 11(1):57–81, March 2003. ISSN 1064-7570. doi: 10.1023/A:1022445108617. URL http://dx.doi.org/10.1023/A:1022445108617.

[70] Heiko Ludwig, Alexander Keller, Asit Dan, Richard P. King, and Richard Franck. Web Service Level Agreement (WSLA) Language Specification, v1.0, January 2003. URL http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf.

[71] Oliver Waeldrich, Dominic Battré, Francis Brazier, Kassidy Clark, Michel Oey, Alexander Papaspyrou, Philipp Wieder, and Wolfgang Ziegler. Ws-agreement negotiation version 1.0. In *Open Grid Forum*, volume 35, page 41, 2011.

[72] Dominic Battré, F. M. T. Brazier, K. P. Clark, M. A. Oey, Alexander Papaspyrou, Oliver Wäldrich, Philipp Wieder, and Wolfgang Ziegler. A proposal for ws-agreement negotiation. In *Proceedings of the 2010 11th IEEE/ACM International Conference on Grid Computing*, pages 233–241. IEEE, October 2010. ISBN 978-1-4244-9348-7.

[73] Stefano Bromuri, Visara Urovi, Maxime Morge, Kostas Stathis, and Francesca Toni. A multi-agent system for service discovery, selection and negotiation. In Carles Sierra, Cristiano Castelfranchi, Keith S. Decker, and Jaime Simo Sichman, editors, *AAMAS (2)*, pages 1395–1396. IFAAMAS, 2009. ISBN 978-0-9817381-7-8. URL http://dblp.uni-trier.de/db/conf/atal/aamas2009-2.html#BromuriUMST09.

[74] Valentina A. M. Tamma, Michael Wooldridge, Ian Blacoe, and Ian Dickinson. An ontology based approach to automated negotiation. In Julian A. Padget, Onn Shehory, David C. Parkes, Norman M. Sadeh, and William E. Walsh, editors, *AMEC*, volume 2531 of *Lecture*

*Notes in Computer Science*, pages 219–237. Springer, 2002. ISBN 3-540-00327-4. URL http://dblp.uni-trier.de/db/conf/amec/amec2002.html#TammaWBD02.

[75] Claudio Bartolini, Chris Preist, and Nicholas R. Jennings. Software engineering for multi-agent systems iii. chapter A Software Framework for Automated Negotiation, pages 213–235. Springer-Verlag, Berlin, Heidelberg, 2005. ISBN 3-540-24843-9. URL http://dl.acm.org/citation.cfm?id=2167504.2167521.

[76] Koen Hindriks, Catholijn M. Jonker, Sarit Kraus, Raz Lin, and Dmytro Tykhonov. Genius: Negotiation environment for heterogeneous agents. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '09, pages 1397–1398, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-0-9817381-7-8. URL http://dl.acm.org/citation.cfm?id=1558109.1558313.

[77] Marco Comuzzi and Barbara Pernici. An architecture for flexible web service qos negotiation. In *EDOC*, pages 70–82. IEEE Computer Society, 2005. ISBN 0-7695-2441-9. URL http://dblp.uni-trier.de/db/conf/edoc/edoc2005.html#ComuzziP05.

[78] Sebastian Hudert, Heiko Ludwig, and Guido Wirtz. Negotiating slas-an approach for a generic negotiation framework for ws-agreement. *Journal of Grid Computing*, 7(2):225–246, Jun 2009. ISSN 1572-9184. doi: 10.1007/s10723-009-9118-3. URL http://dx.doi.org/10.1007/s10723-009-9118-3.

[79] Peer Hasselmeyer, Henning Mersch, Bastian Koller, HN Quyen, Lutz Schubert, and Philipp Wieder. Implementing an sla negotiation framework. In *Proceedings of the eChallenges Conference (e-2007)*, volume 4, pages 154–161, 2007.

[80] Elisabetta Nitto, Massimiliano Penta, Alessio Gambi, Gianluca Ripa, and Maria Luisa Villani. Negotiation of service level agreements: An architecture and a search-based approach. In *Proceedings of the 5th International Conference on Service-Oriented Computing*, ICSOC '07, pages 295–306, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-74973-8. doi: 10.1007/978-3-540-74974-5_24. URL http://dx.doi.org/10.1007/978-3-540-74974-5_24.

[81] J. Yan, J. Zhang, J. Lin, M. B. Chhetri, S. K. Goh, and R. Kowalczyk. Towards autonomous service level agreement negotiation for adaptive service composition. In *2006 10th International Conference on Computer Supported Cooperative Work in Design*, pages 1–6, May 2006. doi: 10.1109/CSCWD.2006.253253.

[82] Karl Czajkowski, Ian T. Foster, Carl Kesselman, Volker Sander, and Steven Tuecke. Snap: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. In *Revised Papers from the 8th International Workshop*

*on Job Scheduling Strategies for Parallel Processing*, pages 153–183, London, UK, UK, 2002. Springer-Verlag. ISBN 3-540-00172-7. URL http://dl.acm.org/citation.cfm?id=646383.689694.

[83] Kwang Mong Sim and Benyun Shi. Concurrent negotiation and coordination for grid resource coallocation. *Trans. Sys. Man Cyber. Part B*, 40(3):753–766, June 2010. ISSN 1083-4419. doi: 10.1109/TSMCB.2009.2028870. URL http://dx.doi.org/10.1109/TSMCB.2009.2028870.

[84] K. M. Sim. Agent-based cloud computing. *IEEE Transactions on Services Computing*, 5 (4):564–577, Fourth 2012. ISSN 1939-1374. doi: 10.1109/TSC.2011.52.

[85] Adriano Galati, Karim Djemame, Martyn Fletcher, Mark Jessop, Michael Weeks, Simon Hickinbotham, and John McAvoy. Designing an SLA protocol with renegotiation to maximize revenues for the cmac platform. In *Web Information Systems Engineering–WISE 2011 and 2012 Workshops*, pages 105–117. Springer, 2013.

[86] Gheorghe Cosmin Silaghi, Liviu Dan Şerban, and Cristian Marius Litan. A framework for building intelligent sla negotiation strategies under time constraints. In *Proceedings of the 7th International Conference on Economics of Grids, Clouds, Systems, and Services*, GECON'10, pages 48–61, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-15680-0, 978-3-642-15680-9. URL http://dl.acm.org/citation.cfm?id=1884547.1884553.

[87] Seokho Son, Gihun Jung, and Sung Chan Jun. An sla-based cloud computing that facilitates resource allocation in the distributed data centers of a cloud provider. *The Journal of Supercomputing*, 64(2):606–637, 2013. URL http://dblp.uni-trier.de/db/journals/tjs/tjs64.html#SonJJ13.

[88] Seokho Son and Sung Chan Jun. Negotiation-based flexible sla establishment with sla-driven resource allocation in cloud computing. In *CCGRID*, pages 168–171. IEEE Computer Society, 2013. ISBN 978-1-4673-6465-2. URL http://dblp.uni-trier.de/db/conf/ccgrid/ccgrid2013.html#SonJ13.

[89] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.*, 25(6):599–616, June 2009. ISSN 0167-739X. doi: 10.1016/j.future.2008.12.001. URL http://dx.doi.org/10.1016/j.future.2008.12.001.

[90] Janki Akhani, Sanjay Chuadhary, and Gaurav Somani. Negotiation for resource allocation in iaas cloud. In *Proceedings of the Fourth Annual ACM Bangalore Conference*, COMPUTE '11, pages 15:1–15:7, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0750-5. doi: 10.1145/1980422.1980437. URL http://doi.acm.org/10.1145/1980422.1980437.

[91] Vladimir Stantchev and Christian Schrpfer. Negotiating and enforcing qos and slas in grid and cloud computing. In Nabil Abdennadher and Dana Petcu, editors, *GPC*, volume 5529 of *Lecture Notes in Computer Science*, pages 25–35. Springer, 2009. ISBN 978-3-642-01670-7. URL http://dblp.uni-trier.de/db/conf/gpc/gpc2009.html#StantchevS09.

[92] Bo An, Victor Lesser, David Irwin, and Michael Zink. Automated negotiation with decommitment for dynamic resource allocation in cloud computing. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, AAMAS '10, pages 981–988, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-0-9826571-1-9. URL http://dl.acm.org/citation.cfm?id=1838206.1838338.

[93] Elarbi Badidi. A framework for software-as-a-service selection and provisioning. *CoRR*, abs/1306.1888, 2013.

[94] Ivona Brandic, Dejan Music, and Schahram Dustdar. Service mediation and negotiation bootstrapping as first achievements towards self-adaptable grid and cloud services. In *Proceedings of the 6th International Conference Industry Session on Grids Meets Autonomic Computing*, GMAC '09, pages 1–8, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-578-9. doi: 10.1145/1555301.1555302. URL http://doi.acm.org/10.1145/1555301.1555302.

[95] Ivona Brandic. Towards self-manageable cloud services. In *Proceedings of the 2009 33rd Annual IEEE International Computer Software and Applications Conference - Volume 02*, COMPSAC '09, pages 128–133, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3726-9. doi: 10.1109/COMPSAC.2009.126. URL http://dx.doi.org/10.1109/COMPSAC.2009.126.

[96] K. Mu Li. All clouds are not created equal. http://insights.wired.com.

[97] Markus C. Huebscher and Julie A. McCann. A survey of autonomic computing degrees, models, and applications. *ACM Comput. Surv.*, 40(3):7:1–7:28, August 2008. ISSN 0360-0300. doi: 10.1145/1380584.1380585. URL http://doi.acm.org/10.1145/1380584.1380585.

[98] Steve R. White, James E. Hanson, Ian Whalley, David M. Chess, and Jeffrey O. Kephart. An architectural approach to autonomic computing. In *ICAC*, pages 2–9. IEEE Computer Society, 2004. ISBN 0-7695-2114-2. URL http://dblp.uni-trier.de/db/conf/icac/icac2004.html#WhiteHWCK04.

[99] Rajkumar Buyya, Rodrigo N. Calheiros, and Xiaorong Li. Autonomic cloud computing: Open challenges and architectural elements. volume abs/1209.3356, 2012. URL http://dblp.uni-trier.de/db/journals/corr/corr1209.html#abs-1209-3356.

[100] Michael J Wooldridge Nicholas R. Jennings. *Agent Technology Foundations, Applications, and Markets*, chapter Applications of Intelligent Agents, pages 3–28. Springer, 1998.

[101] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10:115–152, Wooldridge95.

[102] Peter Stone and Manuela Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, July 2000.

[103] J. Ferber. *Les Systèmes multi-agents: vers une intelligence collective*. I.I.A. Informatique intelligence artificielle. InterEditions, 1995. ISBN 9782729605728. URL http://www2.lirmm.fr/~ferber/publications/.

[104] Ritu Gupta and Gaurav Kansal. A survey on comparative study of mobile agent platforms. *International Journal of Engineering Science and Technology (IJEST)*, 2011.

[105] Florin Leon, Marcin Paprzycki, and Maria Ganzha. A review of agent platforms. Technical report, Multi-Paradigm Modelling for Cyber-Physical Systems, 2015.

[106] M. Dastani A. El F. Seghrouchni J.J. Gomez-Sanz J. Leite G. O'Hare A. Pokahr A. Ricci R.H. Bordini, L. Braubach. A survey of programming languages and platforms for multi-agent systems. *Informatica: An International Journal of Computing and Informatics*, 2006.

[107] Aya Omezzine, Sami Yangui, Narjès Bellamine Ben Saoud, and Samir Tata. Mobile service micro-containers for cloud environments. In *WETICE*, pages 154–160. IEEE Computer Society, 2012.

# Publications List

- Aya Omezzine, Narjès Bellamine Ben Saoud, Said Tazi, Gene Cooperman. Towards a generic multilayer negotiation framework for efficient application provisioning in the cloud. Concurrency and Computation: Practice and Experience, 2017
- Aya Omezzine, Said Tazi, Narjès Bellamine, Gene Cooperman. SLA and Profit-aware SaaS Provisioning through Proactive Renegotiation. The 15th IEEE International Symposium on Network Computing and Applications (IEEE NCA 2016)
- Aya Omezzine, Said Tazi, Narjès Bellamine, Gene Cooperman. Negotiation based scheduling for an efficient SaaS provisioning in the Cloud. The 4th International Conference on Future Internet of Things and Cloud (FiCloud 2016)
- Aya Omezzine, Saïd Tazi, Narjès Bellamine Ben Saoud, Khalil Drira, Gene Cooperman. Towards a dynamic multi-level negotiation framework in Cloud computing. The International Conference on Cloud Technologies and Applications (CLOUDTECH 2015)