OCCI Extension for Autonomic Negotiation in the Cloud Federation

Raouia Bouabdallah[1*], Soufiene Lajmi[2], Khaled Ghedira[1]

[1] Higher Management Institute of Tunis, Tunis University, Bou Choucha Street, Tunisia.
[2] Higher Institute of Computer Science and Multimedia of Sfax, Tunisia.

* Corresponding author. Tel.: +21650933447; email: raouia.bouabdellah@gmail.com
Manuscript submitted September 12, 2018; accepted October 18, 2018.
doi: 10.17706/jcp.13.12.1411-1431

Abstract: Cloud federation is a new paradigm in Cloud Computing. One of the major advantages of this paradigm resides on its ability to help the Cloud Service Provider (CSB) overcome the limitation of their own resources in order to serve their clients. In view of a vast diversity of cloud services being offered by several providers in the cloud federation, it becomes a necessity to couple this paradigm with an automatic negotiation to dynamically form the best possible offer to the clients requirements. In this paper, we propose an extension for the Open Cloud Computing Interface (OCCI) to support the automatic negotiation between different cloud providers using the Contract Net Protocol (CNP) involved in Multi-agent-System (MAS). In order to prove the efficiency and the effectiveness of our approach, we suggest a real case study illustrating how a client can interact with our approach. In addition, we implement a prototype to evaluate the key idea presented in this paper through a set of experiments.

Key words: Cloud Federation, Open Cloud Computing Interface (OCCI), Contract Net Protocol (CNP), Multi-Agent System (MAS).

1. Introduction

Over the last years, the cloud computing has played a major role in the Information Technologies' (IT) evolution. It refers to the provisioning of resources based on pay-as-you-go service model. This model allows a client to pay only computing resources as needed [1]. In the literature, there is no universal definition of cloud computing. It is common that the cloud is characterized by anon demand access to the resources provisioned in an elastic manner. A definition given by NIST [2] is that the cloud provider can be rapidly provisioned and released with minimal management effort through interaction with the provider . It appears from NIST definition that the cloud provider is an essential element of cloud computing. It takes transparently the responsibility of provisioning, managing and maintaining the resources to clients. There is a whole set of commercial cloud providers [3] such as Amazon EC2, Microsoft Azure, Google Compute Engine to name a few, as well as of Cloud Management Platforms (CMPs) in an open cloud ecosystem like OpenNebula, OpenStack, CloudStack, Eucalyptus. Each one of these Clouds provides its own Cloud Resource Management API (CRM-API) so as to allow clients to provision, manage and maintain their cloud resources.

1.1. Problem Statement

The cloud provider faces major problems to assure a correct operation. On the first hand, during peak hours, the cloud provider does not have enough resources to fulfill client's requirements. On the other hand,

during a large amount of time, several resources are not exploited, which lead to a decrease in them profitability. As a result, a new paradigm is required to overcome the lack of the providers' resources by outsourcing the client's demand to other providers within the federation [4]. The cloud providers need to cooperate together in order to provide a new business opportunities such as cost optimization, energy saving and on-demand provision of resources.

Given a plethora of CRM-APIs for the interaction between multiple clients and cloud providers, cloud computing standards are a necessity nowadays to deal with the problem of heterogeneity [5]. Open Cloud Computing Interface (OCCI) is one of the first open extensible standards for managing any kind of resources provided by cloud providers. This standard is supported by a large community of cloud providers including commercial ones such as Amazon EC2 to cite a few, as well as Cloud Management Platforms such as OpenNebula, CloudStack, Eucalyptus. Additionally, this community is implemented by multiple OCCI runtime frameworks , e.g., erocci, rOCCI, pySSF, pyOCNI, and OCCI4Java, which rely on Erlang, Ruby, Python, Java programming languages, respectively [3]. Despite all that, the OCCI standard is defined for an interface of a single cloud installation. Thus, the resources cannot be distributed among different cloud providers which is the case in the cloud federation.

1.2. Approach Overview

In this paper, we propose an extension for Open Cloud Computing Interface (OCCI) to support the automatic negotiation between different cloud providers within a cloud federation using the Contract Net Protocol (CNP). This paper mainly deals with the horizontal federation where the providers can directly cooperate together. The outcomes of this work aim to increase the efficiency and the effectiveness for provisioning cloud resources of an independent cloud provider.

In our previous work [6], we propose an extension for the OVF specification to help the client and the cloud provider describe the required resources in a distributed problem-solving related to the resource allocation. These extensions are found in the area of participants' identification, QoS information and network information. We push further our work and we propose an automatic negotiation approach to dynamically form the best offer between different cloud providers based on the OCCI standard.

In order to add autonomic negotiation facilities to the infrastructure of a cloud provider, we propose new OCCI entities (i.e. Resources and Links) and Mixins. Theses entities allow the cloud provider to dynamically negotiate resources, that response to the client's requirement, from different cloud providers in the federation.

1.3. Paper Structure

This paper is organized as follows: in Section 2, we define the basic aspects related to the cloud federation and the contract net protocol. Then we give an overview on the OCCI standard. Section 3 presents our cloud federation architecture that allows the autonomic negotiation of the cloud resources. Section 4 is the core of our proposition, it describes our extension of the OCCI standard for the autonomic negotiation in the cloud federation. Section 5 describes the use of our OCCI extension to make the autonomic negotiation between different cloud providers. Section 6 implements a prototype to evaluate the efficiency of our approach. In Section 7, we discuss the related works . Finally, section 8 deals with the conclusion and prospects of our work.

2. Background

In this section, we will introduce the different terms related to our work. We start by describing the basic aspects of the cloud federation. Due to the large spectrum of this domain we will limit our description to the negotiation with the contract net protocol. Then, we will introduce the Open Cloud Computing Interface(OCCI).

2.1. Cloud Federation

By looking into the literature, the cloud providers are classified into three categories according to the service models of the cloud computing, which are: Software as a Service (SaaS) provider, Platform as a Service (PaaS) provider and Infrastructure as a Service (IaaS) provider [2]. In this paper, we are interested with the IaaS cloud provider that supplies the material resources enabling clients to create multiple resource configurations for their own applications without being considered the used infrastructure. This provider provides suitable solutions with a high degree of customization to fulfill the different hardware and software requirements of clients.The hardware requirements include processing, memory, storage and network, which are hosted in virtual machines. While, the software requirements include operating systems, tools and applications which ran on virtual machines. Each one of the Clouds provides has its own Cloud Resource Management API (CRM-API) so that the clients enable to provision, manage and maintain their cloud resources. As the client is given the access to only a single cloud provider at a time, the cloud provider is found unable to fulfill a huge number of the client's requirements. Cloud federation is a new paradigm that tackles the issue of integration between different cloud providers to overcome the limited provider resources. For this end, We have proposed in [6] a distributed architecture based on a peer-to-peer cloud federation. This architecture leads to overcome the cloud provider's limited resources and provide a suitable solution with a highly customized environment regarding the software and hardware configuration.We push further this work and we propose an automatic negotiation approach to dynamically provision cloud resources from different cloud providers based on the OCCI standard.

2.2. Contract Net Protocol

The cloud provider has to negotiate with other cloud providers, in the federated environment, and choose the most suitable resources corresponding to the client's requirements to manage the problem of its limited resources. For this purpose, the Contract Net Protocol (CNP) has been used to communicate among the cloud providers in a distributed problem-solving related to the resource allocation. Furthermore, the CNP has been employed to provide an electronic marketplace to buy and sell cloud resources with the best price. It defines a set of rules governing the behavior of providers since the provider's requirements specification, until the requested resources will be issued, in the cloud federation environment. The CNP was standardized by the Foundation for Intelligent Physical Agents (FIPA) [7], in 2002, to be nowadays one of the most used protocols to overcome the task allocation problem in a distributed environment. The CNP is based on the interaction between agents. In FIPA CNP [8], there are two different types of agents: an initiator and a participant. The initiator is an agent that wants to have some task performed by one or more other agents, named participants. It takes the role of a manager wishing to optimize a function that characterizes the task such as the price. For a given task, a set of participants may respond with a proposal and the rest must refuse. Then, the initiator has to continue the negotiation with participants whose proposals are accepted.

2.3. OCCI Specification Overall

Open Cloud Computing Interface (OCCI) is an open standard proposed by Open Grid Form (OGF), for managing any kind of resources. The Open Cloud Computing Interface [9] has been defined as "an abstract of real word resources including the means to identify, classify, associate and extend those resources". OCCI is delivered as a set of specification documents divided into the four categories [9] consisting of the OCCI Core, the OCCI Protocols, the OCCI Renderings and the OCCI Extensions. The OCCI Core specification is an abstraction of real-world resources containing means to extend those resources. The structure of the OCCI Core Model is presented by the UML class diagram shown in Fig. 1. This diagram describes cloud resources as instances of Resource type or a sub-type thereof. One Resource instance cloud be linked and associated with another by an instance of Link type or a sub-type thereof. Resource and Link are a specialization of

Entity. Each Entity instance is identified by a unique Kind instance and may use one or more Mixin instances. Kind is a specialization of Category and may introduce additional capabilities in terms of Actions. An Action identifies an invocable operation applicable to an entity instance. Sub-types of Category are Kind, Mixin and Action. Category instance may have associated Attribute instances whose values characterize a specific Entity instance. The OCCI Protocol specification describes how the OCCI core model can be interacted with a particular protocol (e.g. HTTP, AMQP, etc.). Currently, only the OCCI HTTP Protocol [10] has been specified for the communication between the cloud provider and the client. The OCCI Rendering specification describes a particular rendering of the OCCI Core model independently of the protocol being used. Currently, both OCCI Text Rendering [11] and OCCI JSON Rendering [12] have been specified. As regards the OCCI Extension specifications, every OCCI Extension specification describes a particular extension of the OCCI Core Model. It defines additional Resource, Link, Mixin and Action sub-types for the creation and management of a particular application domain. Currently, three kinds of OCCI Extension specifications have been specified which are OCCI Infrastructure [13], OCCI Platform [14] and OCCI Service Level Agreements [15].

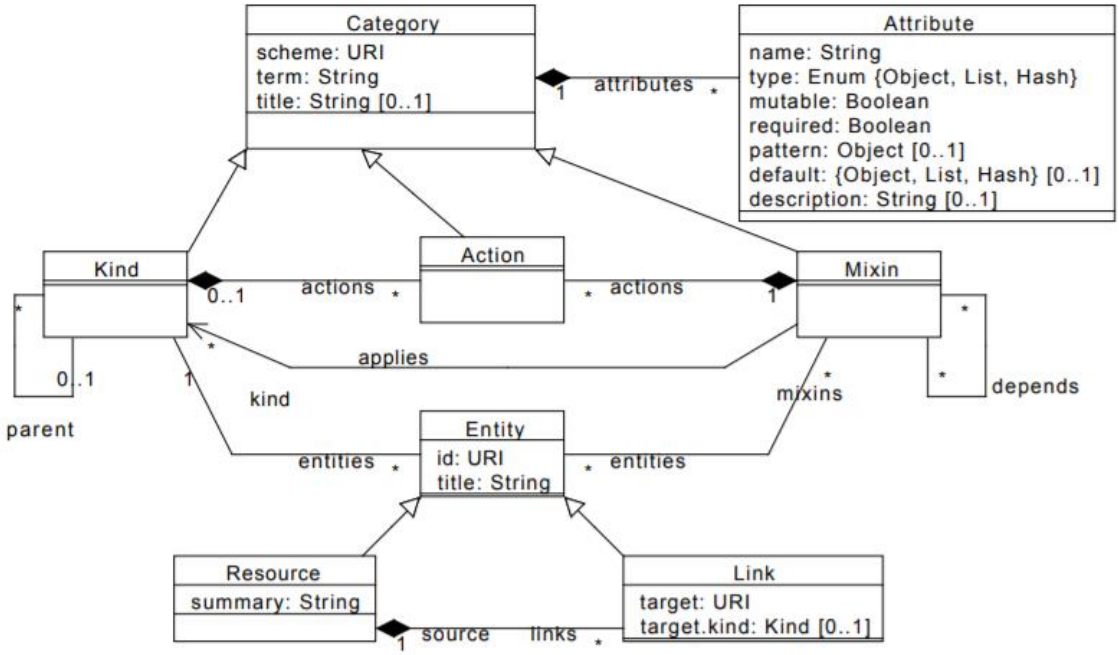

Fig. 1. UML class diagram of the OCCI core model.

3. Automatic Negotiation in the Cloud Federation

In this section, we describe an overviw of our approach to deal with the autonimic negotiation between different cloud providers. Each cloud provider connects with one or more providers that can make the desired resources in the federated environment. This interaction is performed via a Federation Interface (FI). This layer is put on each cloud provider wanting to participate in the cloud federation. As depicted in Fig. 2, the architecture of our approach has two actors interacting through the same interface, which are: the client and the cloud provider. The client represents a person or a cloud provider that needs to deploy his services in a highly scalable environment taking the advantage across the federation and also to benefit from cloud computing characteristics such as on-demand self-service and measured service. To do this, the client firstly browses the *Web portal* of a cloud provider. Secondly, it requests the appropriate resources for deploying his service. Finally, it uses the resources. The cloud provider is the entity that provides physical computing resources as a service to clients in the form of a virtual machine (VM). Federation Interface is

the element that has to be put on each cloud provider that wants to participate in the cloud federation. It complements the common management functionalities that are provided by Cloud Management Platforms (CMPs) by introducing a new layer on it for performing the communication with other cloud providers in the federation. Furthermore, the FI acts as an interface between the client and the provider. Firstly, it receives the resource allocation requestfrom the client. Then, it performs operations in an environment of cloud federation.Finally, it sends the required resource to the client; in which the FI is used to facilitate the negotiation of resources for offering the best fit client requirements in an environment of cloud federation.

Fig. 2 depicts the process implementing the mutualization between different cloud providers. The mutualization process starts when the client (especially the end-user) accesses to our Federation Interface (FI) layer through the *Web portal*. This helps the client introduce his resource allocation request composed of a set of required virtual machines and shows the results. Federation Interface creates an instance of *Management* component relevant to this client and transfers it to the request including the list of the required resources. Management component checks with the Cloud Registry if the demand can be served locally. In case the demand cannot be served locally, the *Management* dynamically creates an instance of *Negotiation* componant to forward the missed resource to other cloud providers in the federation. The autonomic negotiation process begins when the the cloud provider sends a request through the *Negotiation* instance to a set of remote cloud providers via their Federation Interface (FI) layer, so each one of them dynamically creates a *Management* component relevant to this request and forwards to it the request so as to ask for the availability of the resource through the interaction with its Cloud Registry. The cloud provider initiating the negotiation, receives through its *Negotiation* component the proposals received from the different cloud providers answering the call and selects among them the best proposal. Then, the *Negotiation* sends the selected proposal to the Management component which forwards it to the client to confirm it.

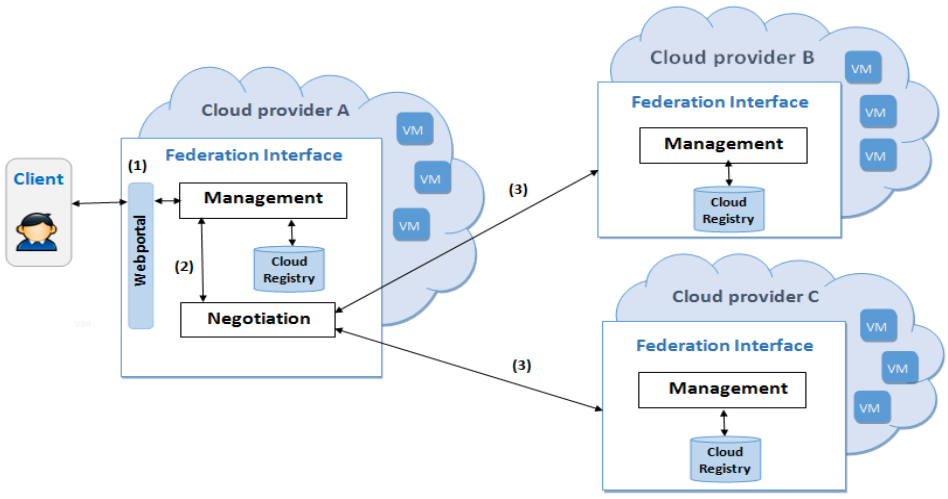

Fig. 2. The proposed autonomic negotiation architecture between cloud providers.

4. OCCI Extension for Autonomic Negotiation in the Cloud Federation

In this section, we define new OCCI entities (i.e. Resources and Links) and Mixins. These last ones allow the cloud provider to establish an autonomic negotiation for provisioning resources, that response to the client's requirement, from different cloud providers in the federation.

4.1. OCCI Resources

We define, in this section, our new Resources inheriting the Resource base type defined in OCCI core [9]

as shown in Fig. 3. Mainly, these resources are used to provide a generic description for the autonomic negotiation resources in the cloud federation. These resources are *Management* and *Negotiation*.

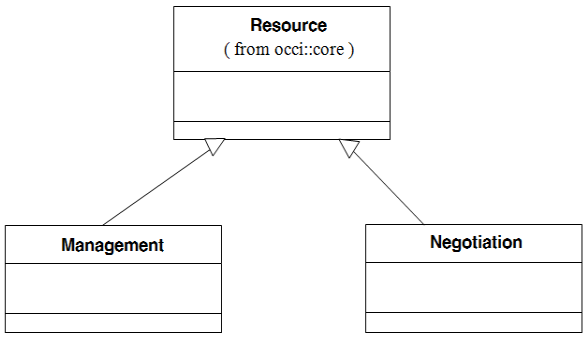

Fig. 3. Resource subclasses for autonimic negotiation in the cloud federation.

4.1.1. Management

The *Management* type represents a generic OCCI Resource. It receives a client's resource allocation request and creates a set of actions to construct the management infrastructure. From a given client request, it is responsible for searching the availability of allocation resources locally and picking among them those that have to be negotiated with the remote cloud providers by the *Negotiation* Resource. *Management* instantiates the needed Entities (i.e., Resources and Links) with the appropriate extensions (Mixins). Since the provider would like to offer first their own resources to the client, the *Management*uses Search Mixin to look locally for the required resources. As shown in Table 1, the *Management*is assigned to the kind instance http://ogf.schemas.sla/occi/management# which has to be exposed by each *Management* instance. Furthermore, it is associated with a set of attributes which are: the name of *Management* instance, the provider, the client and the description.

Table 1. Definition of the *Management*

Model attribute	**Value**			
Scheme	http://ogf.schemas.sla/occi/management#			
Term	Management			
Related	http://ogf.schemas.sla/occi/core#resource			
Attributes	(see below)			
Attributes for the Management				
Name	**Type**	**Mut.**	**Req.**	**Description**
Name	String	No	Yes	Identification the service
Provider	String	No	Yes	Identification the provider
Client	String	No	Yes	Identification the client
Description	String	Yes	Yes	Textual description of the service

Table 2 describes the Actions defined for the *Management* by its Kind instance. *The Management resource* can be started/stopped by invoking the start/stop action.

Table 2. Actions Defined for the *Management* Resource

Action Term	**Target State**
start	active
stop	inactive

4.1.2. Negotiation

The *Negotiation* type represents a generic OCCI Resource that holds all the information related to the negotiation between the cloud providers. It is an abstract resource that could be specified with the appropriate extensions (Mixins).To this end, we introduce the *ContractNetProtocol* and the *BestOffe*r Mixins. Since the negotiation can be described using different protocols (i.e., Contract Net Protocol, Request Interaction Protocol, etc), we use a *ContractNetProtocol* Mixin (detailed in Section 4.3.1) to deal with the contract net protocol. Furthermore, we use *BestOffer* Mixin to specify the strategy that applies on incoming proposals to select the best offer from different cloud Providers. The *Negotiation* type inherits the *Resource* base type defined in the OCCI Core Model. As shown in Table 3, The *Negotiation* is assigned to the kind instance http://ogf.schemas.sla/occi/negotation# which has to be exposed by each *Negotiation* instance. In order to instantiate a Negotiation Resource, we need to specify its attributes which are: the name, the context, the expirationDate. The actions applicable on instances of Negotiation are depicted in the Table 3.

Table 3. Definition of the *Negotiation*

Model attribute	**Value**			
Scheme	http://ogf.schemas.sla/occi/negotiation#			
Term	Negotiation			
Related	http://ogf.schemas.sla/occi/#resource			
Attributes	(see below)			
Attributes for the Negotiation				
Name	**Type**	**Mut.**	**Req.**	**Description**
Name	String	No	Yes	Identification of the negotiation
Context	String	No	Yes	Goal of the negotiation
ExpirationDate	Time (ISO8601)	No	Yes	Date time of the negotiation

The actions applicable on instances of *Negotiation* are depicted in Table 4. The *Negotiation* resource can be started/stopped by invoking the start/stop action.

Table 4. Actions Defined for the *Negotiation* Resource

Action Term	**Target State**
start	active
stop	inactive

4.2. OCCI Link

We define, in this section, our new Links (i.e. *FederationLink*, *NegotiationLink* and *FederationNegotiationLink*) inheriting the Link base type defined in OCCI core [9], as shown in Fig. 4. These links are used to link the specified resources (detailed in Section 4.1) that are dedicated for the autonomic negotiation in the cloud federation.

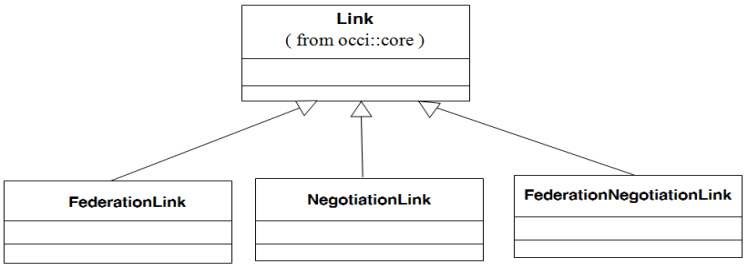

Fig. 4. Link subclasses for autonomic negotiation in the cloud federation.

4.2.1. Federation negotation link

It inherits the Link base type of the OCCI Core Model [9]. This link relates the Negotiation Resource to a specific cloud provider that is able to replay the required demand. It models the transfer of negotiation among different cloud providers. It is an abstract link that could be customized using an instance of NegotiationTool Mixin describing how to pass a negotiation message form one cloud to another in the federation (e.g., REST, XML, XMPP). As depicted in Table 5, The FederationNegotationLink type is assigned to the Kind instance http://schemas.ogf.org/occi/sla/federationneglink# which has to be exposed by each instance of FederationNegotationLink. In order to instantiate a FederationNegotationLink, we need to specify its attributes which are: the name, the context, the initiator and the recipient.

Table 5. Definition of the *Federation Negotiation Link*

Model attribute	**Value**			
Scheme	http://ogf.schemas.sla/occi/federationneglink#			
Term	FederationNegotationLink			
Related	http://ogf.schemas.sla/occi/#link			
Attributes	(see below)			
Attributes for the FederationNegotiationLink				
Name	**Type**	**Mut.**	**Req.**	**Description**
Name	String	No	Yes	Identification the negotiation
Context	String	No	Yes	Goal of the negotiation
Initiator	String	No	Yes	Identification the provider
Recipient	String	No	Yes	Identification the client

The actions applicable on instances of FederationNegotationLink are depicted in Table 6. Each instance could be started or stopped by calling either the start or stop action.

Table 6. Actions Defined for the *Federation Negotation Link*

Action Term	**Target State**
start	active
stop	inactive

4.2.2. Negotiation link

The NegotiationLink inherits the Link base type defined in OCCI Core Model [9]. It connects Management resource with Negotiation resource. It is used to pattern the occurrence of the information related to an unavailable cloud resource hopping to be retrieved through the negotiation with the different cloud providers in the federation. A NegotiationLink is customized with an instance of ConnectionTool Mixin. As depicted in Table 7, the NegotiationLink is assigned to the Link instance http://ogf.schemas.sla/occi/negotationlink# which has to be exposed by each instance of NegotiationLink. In order to instantiate a NegotiationLink, we need to specify its name.

Table 7. Definition of the *Negotiation Link*

Model attribute	**Value**			
Scheme	http://ogf.schemas.sla/occi/negotiationlink#			
Term	NegotiationLink			
Related	http://ogf.schemas.sla/occi/#link			
Attributes	(see below)			
Attributes for the NegotiationLink				
Name	**Type**	**Mut.**	**Req.**	**Description**
Name	String	No	Yes	Identification the negotiation

The actions applicable on instances of *NegotiationLink* are depicted in Table 8. It could be started or stopped by calling either the start or stop action.

Table 8. Actions Defined for the *Negotiation Link*

Action Term	Target State
start	active
stop	inactive

4.2.3. Federation link

The FederationLink inherits the Link base type defined in OCCI Core Model [9]. This Link connects the cloud providers witch each other. It is used to model the occurrence of the information related to an unavailable cloud resource. A FederationLink is customized with an instance of ConnetionTool Mixin. As depicted in Table 9, The FederationLink is assigned to the Link instance http://ogf.schemas.sla/occi/federationlink# which has to be exposed by each instance. In order to instantiate a FederationLink, we need to specify its name.

Table 9. Definition of the *Federation Link*

Model attribute	Value			
Scheme	http://ogf.schemas.sla/occi/federationlink#			
Term	FederationLink			
Related	http://ogf.schemas.sla/occi/#link			
Attributes	(see below)			
Attributes for the FederationLink				
Name	**Type**	**Mut.**	**Req.**	**Description**
Name	String	No	Yes	Identification the negotiation

The actions applicable on instances of FederationLink are depicted in the Table 10. It could be started/stopped by invoking the start/stop action.

Table 10. Actions Defined for the *Federation Link*

Action Term	Target State
start	active
stop	inactive

4.3. OCCI Mixin

We define, in this section, our new Mixins inheriting the Mixin base type defined in OCCI core [9], as shown in Fig. 5. These Mixins specify our new OCCI entities (i.e. Resources and Links) with the appropriate extensions. These mixins are: *ContractNetProtocol*, *NegotiationTool*, *BestOffer*, *Search*, *ConnectionTool and FederationTool*.

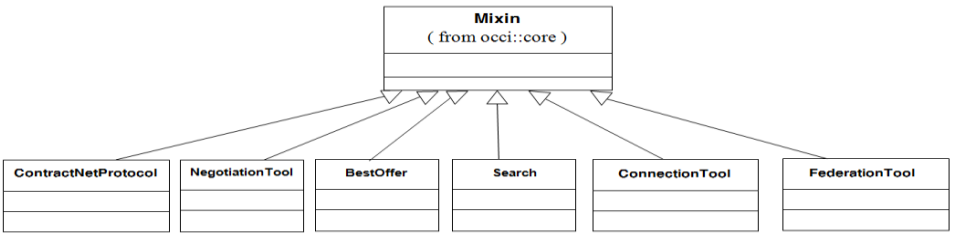

Fig. 5. Mixin subclasses for autonomic negotiation in the cloud federation.

4.3.1. Contract net protocol

Each instance of this Mixin describes the process of negotiation with a specific cloud provider in the federation. It represents an operation applied by the Negotiation Resource to manage the negotiation with a cloud provider using the contract net protocol. This Mixin implements the StrategyInterface that defines a strategy() method. An implementation of this method describes how the Negotiation Resource deals with the contract net protocol. Table 11 defines the actions applied on ContractNetProtocol instances using the contract net protocol.

Table 11. Actions Defined for the *ContractNetProtocol* Mixin

Action Term	Target State
cfp	started
propose	proposal
refuse	refused
accept	accepted
reject	rejected

Fig. 6 illustrates the state diagram for a ContractNetProtocol instance. Each instance of this mixin can be started by invoking the cfp action to send a request to a particular Cloud provider. The cloud provider considers the request and can either propose or refuse it. One of the reasons that this provider might refuse is because it cannot provide the needed cloud resources any more. In this case, this instance of mixin can be refused or have a proposition by invoking the refuse/propose action. Then, it delivers the proposal to Negotiation Resource to choose, among the received proposals provided by the different instances of ContractNetProtocol mixin, which one to be accepted. If the Negotiation Resource picks out the proposal of this mixin instance, this mixin instance is accepted, otherwise it is rejected.

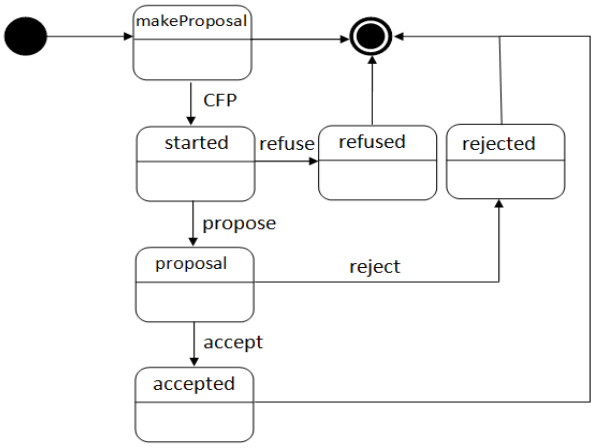

Fig. 6. State diagram for a ContractNetProtocol instance.

4.3.2. Negotiation tool

It implements the needed tools by which negotiation messages are sent to the cloud provider. An instance of the NegotiationTool Mixin enables passing negotiation messages using different mechanisms like REST, XMPP, ACL, XML or HTTP. This Mixin implements a NegotiationTool Interface that defines a negotiate() method specifying the means to exchange the required information for the negotiation. This information may be directly retrieved using the appropriate instance of ContractNetProtocol Mixin. The NegotiationTool Mixin can also refer to an external system of exchanging messages. For instance, an implementation of the negotiate() method carries out an ACL message and sends it to the relevant cloud

provider.

4.3.3. Best offer

This Mixin describes the strategy that is applied on incoming proposals received from different cloud Providers to select among them the best proposal. Technically, this Mixin implements the BestProposalInterface that defines a BestPrice() method. An implementation of this method describes how the Negotiation Resource uses a pricing strategy based on the minimum price in order to sell a service from a cloud provider. This strategy can be set to maximize profitability for each service sold overall.

4.3.4. Search

It provides the useful functionalities to search for the availability of the client's requirements locally. This mixin checks with a storage zone, called CloudRegistry, in case the discovered resources can be served by the cloud provider. The CloudRegistry is responsible for keeping information about allowed types of resources that can be provided as well as keeping information about the cloud provider's capacity available after each resource allocation operation. Technically, this mixin implements a Search Interface that defines a SearchOffer() method to help an instance of Management find the required client resources.

4.3.5. Connection tool

It implements the needed tools by which the Management Resource sends messages to the Negotiation Resource. This Mixin implements a ConnectionTool Interface that defines a connection() method. This method specifies the means to pass the required information (related to the missed resource) to the Negotiation Resource which forwards it to other cloud providers in the federation.

4.3.6. Federation tool

This Mixin describes the needed operation to facilitate the interoperability, especially, in the context of federation between different cloud providers. Technically This Mixin implements a FederationTool Interface that defines a federation() method specifying information concerning the required cloud resources such as processing (compute), memory, virtual disk, operating system and application software.

5. Using OCCI Extension to Negotiate between Cloud Providers

In this section, we explain the use of our OCCI extension (i.e., Entities and Mixins) to make the autonomic negotiation between different cloud providers using the contract net protocol. Fig. 7 depicts the autonomic negotiation's architecture over the cloud providers, which starts when The client (especially the end-user) accesses to our Federation Interface layer. All our OCCI Entities (i.e., Resources and Links) and Mixins are instentiated in this Iayer. It starts by creating an instance of Management Resource relevant to the client and transfers the request to it including the list of the required resources. The Management plays an important role in establishing the required Resources and Links. When the Management is activated, it uses a Search Mixin instance to check if the client demand can be served locally. This Mixin uses the CloudRegistry which keeps information about allowed types of resources and the cloud provider's capacity available after each resource allocation operation. In case the client's demand cannot be served locally, the Management dynamically creates an instance of NegotiationResourceto forward, through the NegotiationLink, the missed resource to other cloud providers in the federation. Whenever this Negotiation is created, the Management starts to negotiate with a set of remote cloud providers via the FederationNegotiationLink. This Link is specified by a NegotiationTool Mixin that defines the means to pass the required information related to the missed resource. The Negotiationuses an instance of ContractNetProtocol Mixin which manages the negotiation process based on the contract net protocol. Each one of cloud provider received a request from another cloud provider, dynamically creates the Management relevant to this request and forwards to this provider a proposal answering its request. At the reception of the proposals, the Negotiation uses a BestOffer Mixin instance to analyze the incoming

proposals received from the different cloud providers and to select among them the best proposal based on the minimum price. Then, the Negotiation sends the best proposal to the Management which forwards it to the client. Finally, the Management receives, from the remoted provider through its FederationLink instance, all the needed information to start the required resources.

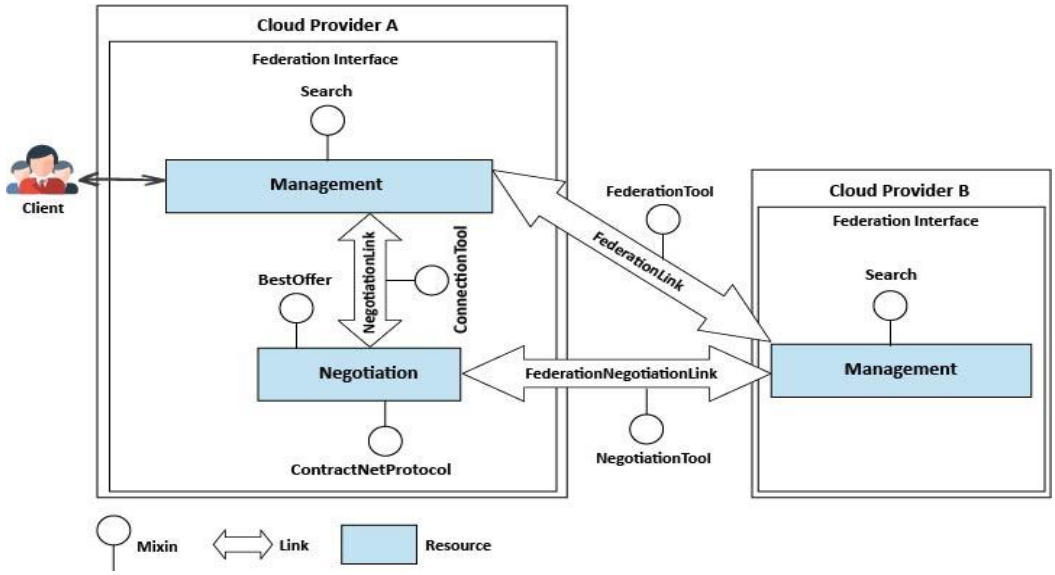

Fig. 7. Autonomic negotiation architecture over the cloud providers.

6. Implementation and Evaluation

In this section, we present a high level view of our implementation that was carried out using a cloud management platform. Then, we evaluate the key ideas presented in this paper to prove the efficiency and the effectiveness of our approach through a set of experiments.

6.1. Implementation

To implement the aforementioned OCCI (Open Cloud Computing Interface) specifications, we used jOCCI [16] project provided by the OGF (Open Grid Forum). This project represents a set of Java libraries implementing the OCCI specifications. The jOCCI project has a jOCCI-core library which covers basic OCCI class hierarchy provided in both OCCI Core [9] and OCCI Infrastructure [13]. This library is used to define classes of Resource, Link and Mixin. Using jOCCI APIs, we extended the Resource class to define our own resources. We also extended the Link and the Mixin classes to implement our links and our mixins respectively.

Every software system is said to be an agent system if it has the following features [17]: knowledge ability, autonomy, learning and communication. An intelligent agent is able to negotiate with other agents through a set of protocols without considering the outside environment. For this end, we used JADE platform (Java Agent Development Framework) [17] to support the Multi-Agent System (MAS) paradigm in our testbed. JADE platform complies with the specifications provided by FIPA which is an IEEE Computer Society standards organization dedicated to promoting the agent-based technology. In our work, to make the communication between different providers and clients, we introduce a new layer called Federation Interface (FI) based on the JADE platform and we design our OCCI Resources as intelligent agents that are able to communicate and control their interactions with each other. We also design our OCCI Links based on the standard FIPA-ACL communication language to support the communication between our different OCCI Resources (eg. agents).

The cloud provider is the entity that provides physical computing resources as a service to clients in the

form of virtual machines. In order to provide the user with the same features that can be found in commercial clouds, we use the OpenNebula platformas as a Cloud Management Platforms(CMP) to perform the operations needed for deploying andmanaging the virtual machines hosted on the IaaS cloud provider. In addition, this platform provides a uniform and homogenous view of virtualized resources regardless of the underlying Virtual Machine Manager (VMM) such as Xen, KVM and WMware.

6.2. Evaluation

We present, in this section, the experiments' environment and the use case of our testbed. Then we describe the experiments' resultsto prove the efficiency and the effectiveness of our approach through three experiments. The first one (Experiment 1) was performed to compare two cloud providers' workload: one using the cloud provider that handles the client's requirements locally and the other using the cloud provider that incorporates our approach of negotiation in the federation. The second one (Experiment 2) was realized to analyze the behavior of the provider operating with our approach in contrast with the traditional one. The last one (Experiment 3) was designed to evaluate the efficiency of our approach to increase the client's demand by providing seemingly unlimited amount of resources across the federation.

6.2.1. Evaluation environment

The physical infrastructure used in this testbed was composed of three cloud providers. Each cloud provider is a server that provides physical computing resources as a service to clients in the form of virtual machines. We have used, in this testbed, OpenNebula platform as a Cloud Management Platforms (CMP) to carry out the operations needed for deploying the virtual machines hosted on each cloud provider. Each CMP was installed on CentOS (Linux) and ran on a node composed of Intel core i7 2.4 GHZ processor with 5 GB of memory and 100 GB of data storage.

6.2.2. Use case

To evaluate our work we present a real use case to illustrate how a client can interact with the cloud provider integrating our approach. The client accesses into the cloud provider through the Web portal provided by Federation Interface (FI) for querying the appropriate resources. Fig. 8 depicts the discovery web page created to specify the client's requirements and to search for their relevant resources. The client fills the fields with the interactive form to specify his relevant virtual machine's configuration. It consists of the software and the hardware configuration. Furthermore, it specifies the resource allocation time and the period of allocation. When the cloud Provider receives the request, it automatically looks for the availability of the required resource through the interaction with the other cloud provider of the federation.

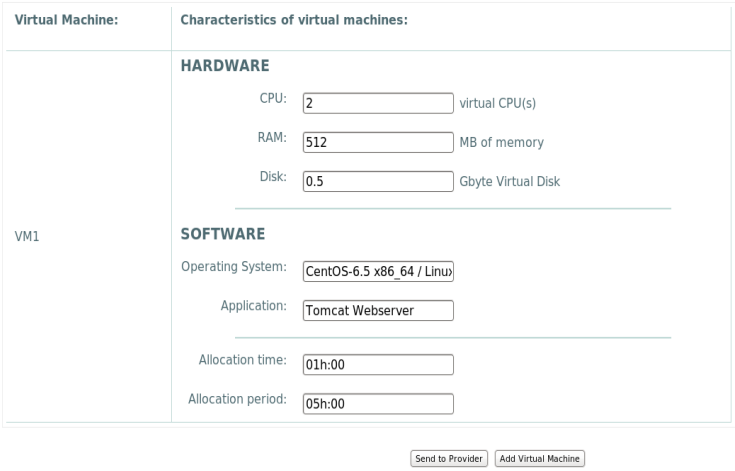

Fig. 8. Web portal of federation interface layer.

6.2.3. Use case instantiation

The input parameters that the clients shall specify for every required virtual machine are depicted in Table 12. These parameters are: (i) the software configuration describes the operating system installed on a virtual machine and the application running on it, (ii) the hardware configuration describes the virtual hardware, such as CPU, memory and storage, (iii) the resource allocation time specifies the time (the unit used is the hour) of starting the allocation of the required resource and (vi) the resource allocation period describes the period required by the client to allocate the relevant resource. To examine our case study, we generate 100 requests of virtual machine vary during 2 days as depicted in Fig. 9.

Table 12. The Values of the Input Parameters in the Cloud Provider

Requests	Software configurations	Hardware configurations	Time of allocation	Period ofallocation
R1	CentOS/ Apacheserver	CPU=0,5 RAM=512 MB Disk=0.5GB	01h:00	03:00
R2	openSUSE/ Tomcat server	CPU=0,5 RAM=512MB Disk=1.0GB	01h:15	05:00
R3	CentOS/ Mysqlserver	CPU=0,5 RAM=512MB Disk=0.5GB	02h:00	04:00
R4	Ubuntu/ Apacheserver	CPU=0,5 RAM=256MB Disk=2.5GB	03h:00	03:00
R5	CentOS/ Tomcatserver	CPU=0.5 RAM=512 MB Disk=1.5GB	04h:30	07:00
R6	Ubuntu/ Apacheserver	CPU=0.5 RAM=512 MB Disk=1.5GB	05h:30	12:00
R7	Fedora/ Apacheserver	CPU=0.5 RAM=512 MB Disk=1.5GB	06h:30	07:00

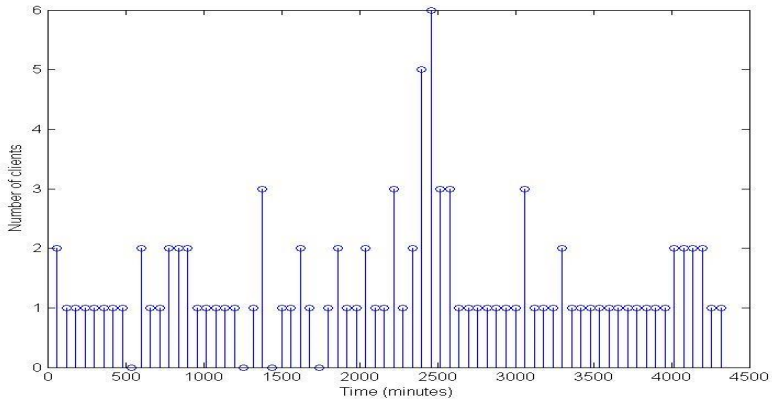

Fig. 9. Number of clients' requests issued per minutes.

6.2.4. Evaluation result

This section describes the results of the mentioned experiments. When the cloud provider receives the client's request, it tries to serve this request. To do so, it checks the availability of the required resources by making the correspondence between the request and the resources locally available. In addition, the cloud provider that incorporates our approach can perform the interaction with other providers to look for the

availability of the resources which satisfies the requirement. The cloud provider shall specify, for each request received, the following output parameters: (i) the software availability describes the availability of the operating system and the application on the required virtual machine. This parameter takes the true value if this condition was reachable by the provider, (ii) the hardware availability describes the availability of CPU, memory and storage. This parameter takes the true value if the cloud provider was able to submit the needed capacity and (iii) the workload of the cloud provider specifies the capacity of workload available on the provider after the allocation of the required virtual machine. Table 13 and Table 14 show the values of the output parameters in the traditional cloud provider and in the cloud provider incorporating our approach, respectively.

Table 13. The Values of the Output Parameters in the Traditional Cloud Provider

Requests	Software configurations	Hardware configurations	Workload of cloud provider
R1	true	true	10%
R2	**false**	true	10%
R3	true	true	20%
R4	true	true	30%
R5	true	true	40%
R6	true	true	50%
R7	**false**	true	50%

Table 14. The Values of the Output Parameters in the Cloud Provider Incorporating Our Approach

Requests	Software configurations	Hardware configurations	Workload of cloud provider
R1	true	true	10%
R2	true	true	20%
R3	true	true	30%
R4	true	true	40%
R5	true	true	50%
R6	true	true	60%
R7	true	true	70%

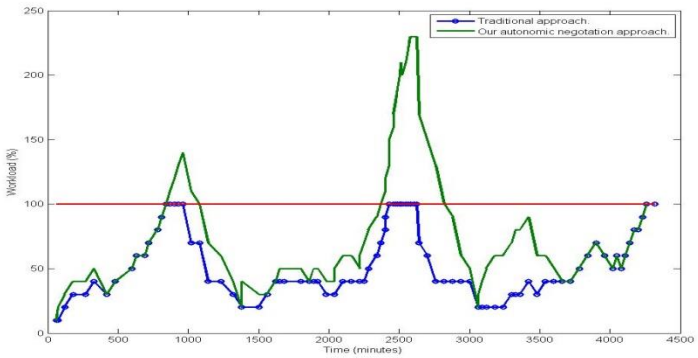

Fig. 10. Cloud providers' workload over time.

Experiment 1: The analyzing of Fig. 10 depicts that the workload of the cloud provider using a traditional resource management approach varies over the times. Sometimes the provider's workload is overused (around 100%), which means its capacity reaches its maximum. This obliges the provider to reject all requests exceeding its maximum capacity as it cannot satisfy their hardware requirements. Furthermore, it loses many clients during the peak hours. This leads so as to decrease the reputation of this provider since the clients would stop trusting it. For this reason, the use of our autonomic negotation approach increases

the provider's capacity, when it is unable to satisfy the requests, by outsourcing resources to external providers in the federation. However, at many other time, the provider's workload is underused (under 50%) which means it has not exploited the entire resources. This under-utilization is justified by two main reasons: First, there are few clients who allocate resources in this period. Second, the cloud provider is not fulfilling the clients' requirementsin case it cannot satisfy their software requirements. Our approach permits this provider to utilize its unused resources by insourcing the providers' requeststo be locally served.

Experiment 2: In this expirement, we analyze the behaviour of the cloud provider incomporating our approach compared with the traditional cloud provider. Our analysis, as shown in Fig. 11, was presented as a function of the number of running virtual machines with their prices over the time. Fig. 11 shows the profitability of the cloud provider for a specific price of 0.09 per VM. The cloud provider using a traditional resource management approach accepts only the requests of virtual machine allocations that are under its capacity. This implicates the loss of the cloud provider's revenue coming from new ones that exceed its capacity and conducts consequently to a reduce in its profit. To avoid this, the use of our autonomic negotation approach may increase the profobility of this provider by outsourcing the virtual machine allocations to external providers when it cannot satisfy their software requirements. Futhermore, our approach allows this provider to insource new providers' requests to be locally served in order to use its unused resources. This option will be profitable for this provider when it obtains revenue from the unused resources. The cloud provider has more profit when its utilization is higher.

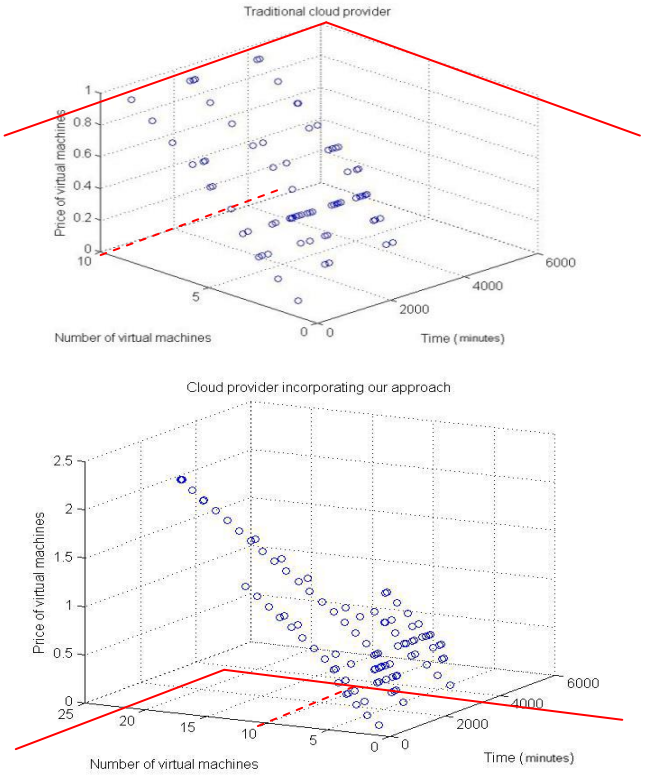

Fig. 11. Cloud provider's behavior over time.

Experiment 3: In this exprimentation, we compare the number of requests served by the cloud provider incorporating ourapproach with an existing provider using a traditional resource management approach. It can be observed in Fig. 12 that the provider using a traditional resource management approach served only 73% from the total clients' sent requests, while the cloud provider incorporating our approach can exceed

98% . The higher number of requests handled by this provider is explained by different policies that our approach applied to allocate resources. In fact, our approach allows the provider that has underutilized resources to rent a part of them to other providers in the federation, in case these resources are already available (insourcing policy). Alternatively, it allows this provider to add the other providers' capacity as its internal one where the client's requests can be served (outsourcing policy). These results indicate that the provider, incorporating our approach, can increase the number of accepted requests, while it preserves its resource allocation's capacity.

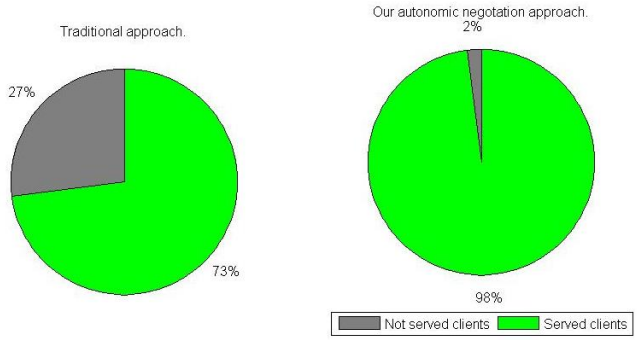

Fig. 12. Satisfied clients.

7. Related Work

In this section, we discuss industrial and academic researchers that share with us the same problematic, vision and goals. these researchers are found in the area of the cloud computing, the cloud federation, the contract net protocol and finally the OCCI extension.

7.1. Cloud Computing

One of the main problems of a cloud provider is the lack of resources to support a huge number of on-demand resources provisioning. In other words, the cloud provider's workload level varies over time. According to [18], there is an over-provision during the day and an under-provision during the night and on weekends. In the over-provision case, the provider has various resources to support all the client's requirements during peak hours. Accordingly, this case causes a large number of underutilized resources which are expensive, most of the time, due to the cost of building data centers and the cost of electric power. The cost of building a data center is related to the purchase of IT equipments, the amount of space and network access required in the data center's facility. According to [19], a data center environment building is based on approximately one thousand to one thousand five hundred dollars per square foot. The energy cost to run the datacenter can be huge. According to the IT analyst Andy Lawrence from the 451 Research Group [20], the operational energy consumption came in at around 30% of the total cost of the infrastructure. We can conclude that this technique is not efficient because it is unable to reduce the costs of resources to be amortized. This obligates the provider to increase its price (for getting Revenues >Costs), which makes it less competitive in the cloud market [18]. In the under-provision cases, the cloud provider does not have enough resources to fulfill the client's requirements. It chooses either to refuse the new client requests or to stop the execution of services already deployed in its resources. This leads the provider to lose its reputation and consequently lose its future clients.

Another problem related to providers is their inability to satisfy all client's requirements according to their own software and hardware specifications which do not necessarily correspond to the client's requirements. They are only interested to provide their resources according to their own ways, for instance,

the types of supported operating systems, the type of the application installed on it, the cost, the performance, etc.

7.2. Cloud Federation

The cloud federation especially the horizontal federation, has received a lot of research attention in the recent years. Emotive Cloud [21], for instance, aims to help the cloud provider take the right decision about the management of its resources. It also allows it to get resources during its peak hours and to rent its underutilized resources by interaction with other cloud providers. We notice that although this work proposes an approach of cloud federation, it is not designed to respond to client's requirements such as the hardware requirement (storage) and the software requirement (the operating system and the applications ran on it). Furthermore, the providers involved in the federation are not totally independent. Each provider can know the status of other providers and evaluate if they have enough resources to host a given virtual machine by exposing the internal behavior of the providers, which is inadequate. By Inter-Cloud [22], this work allows the cloud provider to enhance its performance by leveraging resources from multiple clouds. So, the cloud provider accesses the *Cloud Exchange* to publish the characteristics of its available virtual machines and discover other providers within a cloud federation. We note that although this work proposes a cloud federation approach to support the negotiation of cloud resources, the cloud providers cannot negotiate directly without interaction with the *Cloud Exchange* which plays the role of the broker between providers. In addition, the description of the resources, in this work, is limited to the hardware's characteristics of the available virtual machines (such as the the amount of memory and the number of cores). It does not take into consideration the operating system and the application installed on it. OPTIMIS [23] is one of the important projects dealing with cloud federation. It proposes a platform to enable a flexible provisioning of the cloud service. This platform manages the three fundamental steps of the service life cycle: the construction, the deployment and the execution of the service. The key process, during the deployment step, is the negotiation of SLA (Service Level Agreement) between the service provider and the infrastructure provider. However, it ought to be noted that this negotiation is performed manually.

7.3. Contract Net Protocol

The communication between cloud providers, in this work [24], follows the *Alternate offers protocol* inspired from the contract net protocol to buy and sell virtual machines. However, the communication between the cloud providers is based on web services. These web services may be useful for the communication among elements of the cloud federation, but this communication is not dynamic. It requires the cloud provider to interact, each time, with other providers to agree, modify or reject the offer. In [22], the authors propose an agent-based cloud computing to negotiate and compose cloud resources. This work adopts a contract net protocol to select cloud resources to satisfy the client's requirements. However the communication with the client and the cloud provider is indirect. It is performed by brokers that compose multiple resources from different providers, and pack them to the client as an unified service.

7.4. OCCI

OCCI SLA Model [12] is an extension of the OCCI Core Model 1.2. This specification is used to create and manage SLA related resources. The creation and management of SLAs include three phases, which are: the negotiation, the agreement and the execution. The negotiation phase is established when the consumer submits an offer to the cloud provider. In the agreement phase, the cloud provider can decide whether to accept or to reject the offer. It is also possible to provide a counter-offer to the customer. The execution phase happens when the offer gets accepted. OCCI SLA Mddel extends the OCCI Core Model withtwo new types, which are: *Agreement* and *AgreementLink*. The *Agreement* type is a generic contract resource

between a customer and acloud provider, which is used for negotiation, provisioning and monitoring cloud resources. The *AgreementLink* represents a link that associates an agreement instance with the provisioned resources after the customer and the cloud provider have agreed to the context of the agreement. However, the OCCI SLA specification has its own drawbacks: First, this specification is defined to establish an interface for the creation and the management of resources in the single cloud provider. Therefore, cloud resources cannot be distributed among different cloud providers which is the case in the cloud federation architecture. Second, this specification defines two entities (*Agreement* and *AgreementLink*) for representing a basic information about the SLA life-cycle. Thus, these entities cannot be used to support the whole life-cycle of negotiation between the different parties.

A. Stanik *et al.* [25] propose an advanced specification for the OCCI SLA Model. Moreover, It defines new entity types for distributing the SLA and monitoring resources over different cloud providers such as *Offer* and *Negotiation*. The client discovers the required offer hosted at the broker Inter-cloud root which deploys the whole offer provided by the providers. Additionally, the client is able to negotiate the offer, discovered before, with the required provider under specific terms. Although, this work handled the problem of distributing resources among different cloud providers within inter-cloud architecture, this proposed architecture is based on the concept of mediation or brokering which is a global market place for cloud services, where all clients can discover cloud services of independent providers. Moreover, the *Negotiation* type proposed in this multi-cloud architecture is established to handle the negotiation between just one client and one provider. This *Negotiation* instance has not been established to provide the best cloud service, which satisfies the client'srequirements, among different cloud providers, as in our case.

The FP7 mOSAIC [26] proposes an extension of OCCI specification for the cloud-infrastructure provider. This extension is conceived by Cloud Agency, which is a broker based on the multi-agent system. This broker accessesthe different cloud providers, on behalf of the client, to pick up among themone cloud service. Then the client negotiates with the appropried provider about the selected cloud service by using the FIPA contract-net protocol. We notice that, although this work proposes an extension of OCCI specification, this extension is not done by introducing new Resource and Link types as well as Mixin type of the OCCI Core Model, as in our case. This work is interested only to verify the correctness of a client's OCCI Rest request and how to translate this request to the ACL message. Additionally, the whole lot of client's requests delivered to the broker leads to decrease its ability to process these requests when their number increases. So the presence of bottleneck [26]. The outcomes of our work aim to increase the efficiency end the effectiveness of provisioning the cloud services of an individual provider.

8. Conclusion

In this paper, we have proposed an extension for Open Cloud Computing Interface (OCCI) to support the automatic negotiation between different cloud providers using the Contract Net Protocol (CNP).Our extension respects the last version of the OCCI standard.Furthermore, we have explained the use of our OCCI extension (i.e., Entities and Mixins) to make the autonomic negotiation architecture over the federated cloud providers. To evaluate our work we have presented a real use case to demonstrate the application of our autonomic negotiation approach in the cloud federation. We have described also a set of experiments to prove the efficiency and the effectiveness of our approach.

In our future work, we will extend our automatic negotiation approach to dynamically form the best possible contractual agreement between different parties. This agreement is based on Service Level Agreement (SLA) to respond to the clients' requirements. It keeps the information about a resource that has either been negotiated with an instance of Negotiation or been provided locally by cloud provider.

References

[1] Khedr, A., & Idrees, A. (2017). Enhanced e-learning system for e-courses based on cloud computing. *JCP, 12,* 10-19.

[2] Liu, F., Tong, J., Mao, J., Bohn, R., Messina, J., Badger, L., & Leaf, D. (2011). *National Institute of Standards and Technology Special Publication,* September.

[3] Merle, P., Barais, O., Parpaillon, J., Plouzeau, N., & Tata, S. (2015). A precise metamodel for open cloud computing interface. *Proceedings of IEEE 8th International Conference on Cloud Computing*. New York.

[4] Jinlai, X., & Balaj, P. (2018). Optimized contract-based model for resource allocation in federated geo-distributed clouds. *IEEE Transactions on Services Computing.*

[5] Sutherland, S., & Chetty, G. (2016). Investigation into interoperability in cloud computing: An architectural model. *JCP, 11,* 159-168.

[6] Bouabdallah, R., Lajmi, S., & Ghedira, K. (2016). Resources provisioning within cloud federation. *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (pp. 003978-003983). Budapest.

[7] Bellifemine, F., Poggi, A., & Rimassa, G. (2001). Developing multi-agent systems with JADE. *Intelligent Agents VII Agent Theories Architectures and Languages,* 89-103.

[8] FIPA. (2002). *Fipa Contract Net Interaction Protocol Specification.*

[9] Nyren, R., Edmonds, A., Papaspyrou, A., Metsch, T., & Parak, B. (2016). Open cloud computing interface -core. *Recommendation GFD-R-P.221, Open Grid Forum.*

[10] Nyren, R., Edmonds, A., Metsch, T., & Parak, B. (2016). Open cloud computing interface — Httpprotocol. *Recommendation GFD-R-P.223, Open Grid Forum.*

[11] Edmonds, A., & Metsch, T. (2016). Open cloud computing interface — Text rendering. *RecommendationGFD-R-P.229, Open Grid Forum.*

[12] Nyren, R., Feldhaus, F., Parak, B., & Sustr, Z. (2016). Open cloud computing interface — Jsonrendering. *Recommendation GFD-R-P.226, Open Grid Forum.*

[13] Metsch, T., Edmonds, A., & Parak, B. (2016). Open cloud computing interface — Infrastructure. In: Recommendation GFD-R-P.224, Open Grid Forum.

[14] Metsch, T., & Mohamed, M. (2016). Open cloud computing interface — Platform. *RecommendationGFD-R-P.227, Open Grid Forum.*

[15] Katsaros, G. (2016). Open cloud computing interface — Service level agreements. *RecommendationGFD-R-P.228, Open Grid Forum.*

[16] Kimle, M., Parak, B., & Sustr, Z. (2015). jOCCI — General-purpose OCCI client library in Java. *Proceedings of International Symposium on Grids and Clouds (ISGC).* Taipei, Taiwan.

[17] Sandita, A. V., & Popirlan, C. L. (2015). Developing a multi-agent system in jade for information management in educational competence domains. *Procedia Economics and Finance*, *23,* 478-486.

[18] Goiri, I., Guitart, J., & Torres, J. (2012). Economic model of a cloud provider operating in afederated cloud. *Information Systems Frontiers*, *14,* 827-843.

[19] *Data Center Construction Costs.* Retrieved from http://www.datacentrerealty.com/data-center-construction-costs-2/

[20] Shead, S. (2013). *Energy Costs Mean Tough Decisions for Datacentre Owners.*

[21] Calheiros, R. N., Toosi, A. N., Vecchiola, C., & Buyya, R. A. (2012). Coordinator for scaling elastic applications across multiple clouds. *Future Generation Computer Systems, 28,* 1350-1362.

[22] Calheiros, R. N., Toosi, A. N., Vecchiola, C., & Buyya, R. (2012). A coordinator for scaling elastic applications across multiple clouds. *Future Generation Computer Systems*, *28(8),* 1350-1362.

[23] Zsigri, C., Ferrer, A. J., Barreto, O., Sirvent, R., Guitart, J., Nair, S., Sheridan, C., Djemame, K., Elmroth, E., & Tordsson, J. (2013). *Why Use OPTIMIS ?*

[24] Calheiros, R. N., Toosi, A. N., Vecchiola, C., & Buyya, R. (2012). A coordinator for scaling elastic applications across multiple clouds. *Future Generation Computer Systems, 28(8),* 1350-1362.

[25] Stanik, A. (2016). *Thesis: Service Level Agreement Mediation, Negotiation and Evaluation for Cloud Services in Intercloud Environments.*

[26] Venticinque, S., Tasquier, L., & Martino, B. D. (2012). Agents based cloud computing interface for resource provisioning and management. *Proceeding of 2012 Sixth International Conference on Complex, Intelligent, and Software Intensive Systems* (pp. 249-256).

Raouia Bouabdallah is currently a Ph.D student from the Higher Institute of Management University of Tunis and a member of the research laboratory SSOIE-COSMOS. Her research interest involves optimization of resources allocation in the cloud federation, scaling cloud applications and performance in the cloud. She received her computer engineer degree from ISIMS (Tunisia) in 2010. She also obtained her master degree from ISIMS (Tunisia) in 2012. Contact her at raouia.bouabdellah@gmail.com.

Soufiene Lajmi has obtained the Ph.D degree in computer science from ENSI (Tunisia) and UCBL (France) in 2009. He also has received the engineer degree in computer science from ENSI (Tunisia) in 2004, the master degree from ENSI (Tunisia) in 2005. He is a member of the research laboratory SSOIE-COSMOS. His research areas include web services, multi-agents systems and cloud computing. Contact him at soufiene.lajmi@ensi.rnu.tn.

khalid Ghedira has received the engineer degree in hydraulic from ENSEEIHT (France), the specialized engineer degree in computer science and applied mathematics from ENSIMAG (France), the master degree then the Ph.D degree in artificial intelligence from ENSAE (SupAero France). He also obtained the habilitation degree in computer science from ENSI in 1999 (Tunisia). He was fellow research at the Institute of Computer Science and Artificial Intelligence of Neuch tel (Switzerland 1992-96) and expert consultant at British Telecom (England 1995). He also was the head of ENSI (2002-08). He is currently a professor at the Higher Institute of Management (ISG Tunis) and the president of both the Tunisian Association of Artificial Intelligence (ATIA) and the Intelligent Information Engineering Laboratory (LI3 Tunisia). Professor Ghedira has co/authored more than 200 journal and conference research papers. He has also written two text books on respectively metaheuristics and production logistics. His research areas include multi-agents systems, constraint satisfaction problems, scheduling and transport problems, multi-criteria decision making and metaheuristics. He is member of several program committees relative to various conferences and journals and has also co/organized several International conferences.