# SLA NEGOTIATION AND BROKERING FOR SKY COMPUTING

Alba Amato, Loredana Liccardo, Massimiliano Rak and Salvatore Venticinque
*Dipartment of Information Engineering, Seconda University of Naples, Aversa, Italy*

Keywords: SLA, Negotiation, Cloud, Agents.

Abstract: Cloud computing represents an opportunity for IT users to reduce costs and increase efficiency providing an alternative way of using IT services. Elastic provisioning plays an important role by giving the possibility to get the best resources configuration that satisfies the application requirements. Even if there are many Cloud Providers, with a rich and various offer of technological solutions, above all at IAAS, however there is neither support of SLA negotiation, nor of SLA management yet. One of the most important issue in such a context is the lack of negotiation interfaces and mechanisms by current providers for dynamic provisioning, which instead only make available a configuration form to submit the request for a specific resource among the available ones. In the following we propose the design of a tool for SLA based dynamic provisioning of Cloud Resources at IAAS, that offer to the user negotiation and brokering facilities by integrating multiple models. We present a prototype implementation of our architecture using the mOSAIC framework.



## 1 INTRODUCTION

Cloud computing represents an opportunity for IT users to reduce costs and increase efficiency providing an alternative way of using IT services. In this scenario elastic provisioning plays an important role by giving the possibility to get the best resources configuration that satisfies the application requirements. Even if there are many Cloud Providers, with a rich and various offer of technological solutions, above all at IAAS, however there is neither support of SLA negotiation, nor of SLA management yet. This is due to two main issues. The first problem concerns a semantic gap between the application requirements at user side and the service level at IAAS. This gap deals with the performance parameters that describe the application (e.g: memory usage, throughput, turnaround, ...) and the ones which characterize the Cloud resource at IAAS (e.g: CPU architecture, main memory, number of cores, disk space ...). This issue is addressed by defining a mapping between high level to low level performance indexes (C Emeakaroha et al., 2010), but it is out of the scope of this paper. The second issue is the lack of negotiation interfaces and mechanisms by current providers for dynamic provisioning, which instead only makes available a configuration form to submit the request for a specific resource among the available ones. In fact, the absence of standards for Cloud solutions that ensure portability among different vendors' technologies represents an additional barrier for the development of such kind of services. To overcome these problems it is fundamental to have a general purpose set of specifications and facilities for cloud that are vendor independent, platform neutral. A first proposal of standard in Cloud is represented by OCCI (Open Cloud Computing Interface) that is a protocol and API for management tasks. This solutions is aimed at the fulfillment of three requirements: integration, portability and interoperability for common tasks including deployment, autonomic scaling and monitoring still offering a high degree of extensibility[1]. Here we focus on a solution that is ready to work now, for overcoming the current limitation of the Cloud offer, that does not allow the users to access the Cloud market by a business interface. In the following we propose the design of a tool for SLA based dynamic provisioning of Cloud Resources at IAAS, that offer to the user negotiation and brokering facilities by integrating multiple models. We present a prototype implementation of our architecture using the mOSAIC framework.

## 2 RELATED WORK

In this section, we briefly describe several related works and classify their approaches for concurrent resource negotiation with multiple providers. In par-

[1] www.occi.org

Amato A., Liccardo L., Rak M. and Venticinque S..
SLA NEGOTIATION AND BROKERING FOR SKY COMPUTING.
DOI: 10.5220/0003964406110620
In *Proceedings of the 2nd International Conference on Cloud Computing and Services Science* (CloudSecGov-2012), pages 611-620
ISBN: 978-989-8565-05-1

ticular we focus our attention on protocols that are the set of rules that govern the interaction defining the types of entities that can participate in the negotiation, the status, the events that cause the change in status and the correct actions of the participants in particular situations. The contributions discussed propose negotiation protocols, which are extension of either Contract Net Protocol or Rubstein's alternating offers protocol. Aknine et al. (Aknine et al., 2004) propose an extended version of the contract-net protocol (CNP) to support concurrent many to many negotiation processes enabling an agent to manage several negotiation processes in parallel. This approach introduces a two phase negotiation process. In the proposal exchange phase both sides exchange proposals/counter-proposals until one agent sends Pre-Accept/Pre-Reject to the other. In the proposal formalization phase an agent sends its formal proposal if it is pre-accepted, or the counter-proposal otherwise. Besides this approach optimizes the length of the negotiation processes among agents, reduces the contractors' decommitment situations, enables the detection of failures of an agent participating in a negotiation process and prevents a negotiation process with blocked agents. Chhetri et al. (Chhetri and al., 2006) present a multi-agent framework with a two-layered architecture, that combines the alternate offer and the CNP protocols. In this approach a coordinator agent is used: to decompose the negotiation into individual service utilities that are forwarded to the individual negotiation agents that negotiate with multiple providers iteratively over multiple rounds, to coordinate the negotiation for different services and to analyze the service composition with the aim of mapping the QoS requirements of the user request into the QoS requirements of all services within the composition selecting the best combination.

Siebenhaar et al. (Siebenhaar and al., 2011) propose an extension of CNP approach that combine services from multiple providers negotiating individual QoS parameters across multiple tiers using a two-phase protocol. The composition of services is coordinated and managed by a Coordinating Entity that in turn creates several Negotiating Entities according to the number of services in the composition request with the task to negotiate concurrently with multiple providers over the QoS parameters of a particular service. In the second phase of the protocol, multiple overbidding is allowed and time-dependent strategies, which stop the overbidding process if necessary, are applied. Sim and Shi (Sim and Shi, 2010) propose an approach based on Alternate Offers protocol in which there is a management of multiple one-to-many concurrent negotiations between a user and multiple resource providers, and of intermediate contracts between users and providers that can apply a time-dependent strategy and can renege on a contract during the negotiation by paying a penalty fee. Dang and Huhns (Dang and Huhns, 2006) propose an extension of the alternate offers protocol in order to support several negotiation processes in parallel in case of many-to-many negotiations. Sim (Sim, 2010) proposes an extension of the alternate offers protocol that supports multiple complex negotiation activities in interrelated markets between user agents and broker agents, and between broker agents and provider agents. In our approach, given the nature of the problem, the negotiation is one-to-one. In particular we have two kind negotiations:

- The brokering managed by cloud agency among the available offer from supported providers;
- The negotiation between users and SLA applications.

For the management of brokering is used an implementation of Contract-Net Protocol for a dedicated cloud application while for the management of SLA negotiation we develop components that enable us to implement any model of strategic trading is us. As an example, in this paper, is used an extension of Contract-Net Protocol that allows the user to build the offer by selecting the favourite proposals.

## 3 SLA NEGOTIATION IN THE SKY COMPUTING PARADIGM

In order to provide a solution for the SLA negotiation at IAAS in the current Cloud context we conceived the architectural model that is shown in Figure 1. The negotiation problem is addressed here in two steps, by two independent components, which interact according to a well defined protocol:

- A Cloud Application supports the implementation of many negotiation models. It allows the user to select the requirements to be negotiated and the model of negotiation to be used.
- An autonomic Cloud broker (Cloud Agency) accepts an SLA template and a set of brokering rules, by which it provides the best proposal for that request.

The SLA negotiation allows the user to access the Cloud market by an high level business interface. The user can choose among the supported negotiation models. The negotiation interface allows to delegates a lot of stuff to the application and to focus

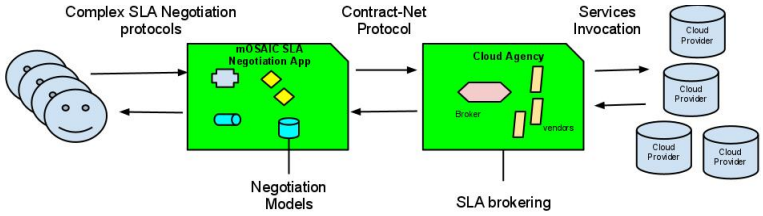


Figure 1: Architectural Model for SLA Negotiation.

on the desired solution in terms of requirements, service levels and business parameters. The application supplies also to the lack of mechanisms for dynamic resource provisioning at provider side, by implementing a complex logic, that is reconfigurable and policy driven. A this level the user is going to define high level and low level requirements which are necessary for running his Cloud applications. Here we do not provide high level to low level mapping, hence only low level parameters can be defined. For this purpose we identified three sets of information: i) mandatory requirements, ii) negotiation model and negotiation parameters, iii) brokering rules and parameters. The retrieval of proposals and the brokering among them, by the rules defined at iii) is delegated to Cloud Agency. The provisioning service by Cloud Agency is used to get all proposals, which are necessary as an input for the SLA negotiation implemented at application level. Cloud agency provides a vendor-agnostic interface to a number of providers and technologies available in the Cloud market. It implements a very simple negotiation protocol, characterized by a limited interaction, that is described in the following section. This means that Cloud Agency will make a first choice among the retrieved proposals which are compliant with the mandatory requirements i), and according to some brokering rules and those parameters, which do not need to be negotiated. The brokering solution enable to find the best offerings respect to a given set of chosen parameters and rules, in the context of the negotiation, instead, the goal is to offer to the final user more than a single choice, so that it is possible to involve him in the final decision about the resources to be acquired. An example for Cloud negotiation could look for a Virtual Machine with i) specific CPU architecture and a fixed amount of memory, ii) any number of cores to be negotiated, iii) brokering the best price among the proposals which satisfy i) and ii). The cloud application has an intermediary role between the user and the broker, implementing the negotiation protocol toward the users and the brokering protocol toward the cloud agency and offers both a REST-based interface for enabling the SLA negotiation and a Web interface that embeds the REST requests. SLA are represented trough the WS-Agreement standard, even if the negotiation protocols are not standard compliant, in fact Ws-Agreement does not offer big support in terms of Negotiation protocols and it is Web Services based, while we need a REST interface. The Cloud Application we propose is developed in order to be able to scale with the number of parallel requests, moreover it should be able to flexibly support new kind of protocols.

# 4 SLA BROKERING: CLOUD AGENCY

Cloud Agency is a multi agent system (MAS) that accesses, on behalf of the user, the utility market of Cloud computing to maintain always the best resources configuration that satisfies the application requirements. This system is being designed and developed within the research activities of the FP7 mOSAIC project. It is in charge to provide the collection of Cloud resources, from different vendors, that continuously meets the requirements of users applications. According to the available offers, it generates a service level agreement that represents the result of resource brokering and booking with available providers. The user is able to delegate to the Agency the monitoring of resource utilization, the necessary checks of the agreement fulfillment and eventually re-negotiations. Cloud Agency will supplement the common management functionalities which are currently provided by IAAS Private and Public infrastructure with new advanced services, by implementing transparent layer to IAAS Private and Public Cloud services;

Cloud Agency will support the Cloud user in two different scenarios:

- **Deployment:** to discover and buy the available resources needed to run Cloud applications,
- **Execution:** to monitor and eventually to reconfigure Cloud resources according the changed requirements of the Cloud Application.

Here we focus on the first one by which Cloud resources at IAAS are brokered among the available

proposal from many heterogeneous providers.

## 4.1 An OCCI Extended Model for Cloud Provisioning

Obtaining a common interface is the focus of the OCCI group that has defined a model for Cloud management at IAAS. OCCI defines entities, API and protocol. The specification of Cloud API is a Resource Oriented Architecture (ROA) that uses Representational State Transfer (REST)protocol. The OCCI core meta-model (Nyr n et al., 2011) provides means of handling abstract Cloud resources. The heart of the OCCI Core Model is the Resource type. Any resource exposed through OCCI is a Resource or a sub-type thereof. A resource can be e.g. a virtual machine, a job in a job submission system, a user, etc. The Resource type contains a number of common attributes that Resource sub-types inherits. The Resource type is complemented by the Link type which associates one Resource instance with another. The Link type contains a number of common attributes that Link sub-types inherit. Entity is an abstract type, which both Resource and Link inherit. Each sub-type of Entity is identified by a unique Kind instance. The Kind type is the core of the type classification system built into the OCCI Core Model. Kind is a specialization of Category and introduces additional resource capabilities in terms of Actions. An Action represents an invocable operation applicable to a resource instance. The last type defined by the OCCI Core Model is the Mixin type. An instance of Mixin can be associated with a resource instance, i.e. a sub-type of Entity, to "mixin" additional resource capabilities at run-time. To be compliant with the OCCI standard it is necessary to extend the core model using inheritance at two points: mixin: to provide IAAS resources with monitoring and provisioning capabilities and entity: to represent those new concepts, which are introduced by the services we are going to describe. Cloud Agency extends the OCCI model by autonomic and dynamic services, it is necessary that they new functionalities can be deployed in user's Cloud. In Figure 2 the new entities and mixin, an their relationship with the OCCI core model are shown. Among the entities, a Call for Proposal (CFP) describes the list of resources which are necessary to run a Cloud application. A Cloud customer defines the values of those attributes of OCCI Compute, Storage and Links which are significant for his application requirements. A CFP will include also the negotiation rules to select the best offer among the ones proposed by providers. A Proposal is a candidate for a Service Level Agreement (SLA). It includes an instance for each resource in the

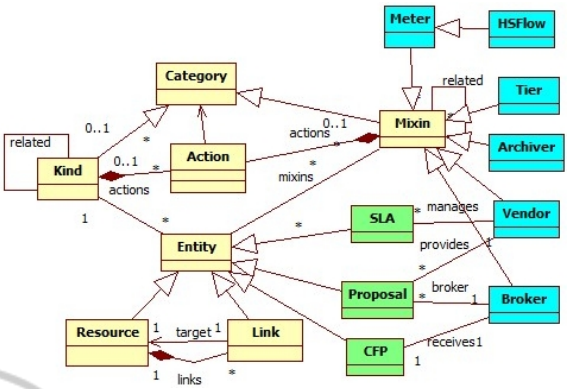


Figure 2: OCCI Extended model.

CFP with all the attribute specified, including other information such as the cost and service levels. It can be accepted or refused by the customer. An SLA is an accepted proposal, it is agreed by the customer with one provider. Among the new mixin, a Vendor is committed to get an offer for resource provisioning, to accept or refuse that offer, to get information about a resource or to perform an action on it against a specific Cloud provider or technology. A Broker is an intermediary that provides information and assists in finding the right cloud-based solution. It receives a CFP, asks to vendors for available offers, brokers the proposal that best serves the user and allows to close the transaction. The Meter is a mixin that monitors some parameters chosen by user and gives the measure of performance indexes for those parameters. It can be deployed into the Cloud, within a Virtual Machines, to dynamically complement the user's Cloud resource with this new capability. We need to run Meter instances on any resource to perform locally specialized algorithm to take measures. An Archiver collects measures from Meters and stores them in a knowledge base. It provides metrics and performance statistics computed on the knowledge base. A Tier is a mixin that use statistics provided by the Archiver, it detects critical conditions such as SLA violations, resource saturation or underutilization and then informs the user or autonomically performs required reaction as new provisioning or management.

## 4.2 Cloud Agency: Static View

According to our extended OCCI model Cloud Agency is both a mixin itself and a collection of those mixins presented before. It can run anywhere, also in an user's Resource, if it has been implemented as a deployable software component. Cloud Agency has been implemented as a Multi Agent systems. The different components of the agency architecture is

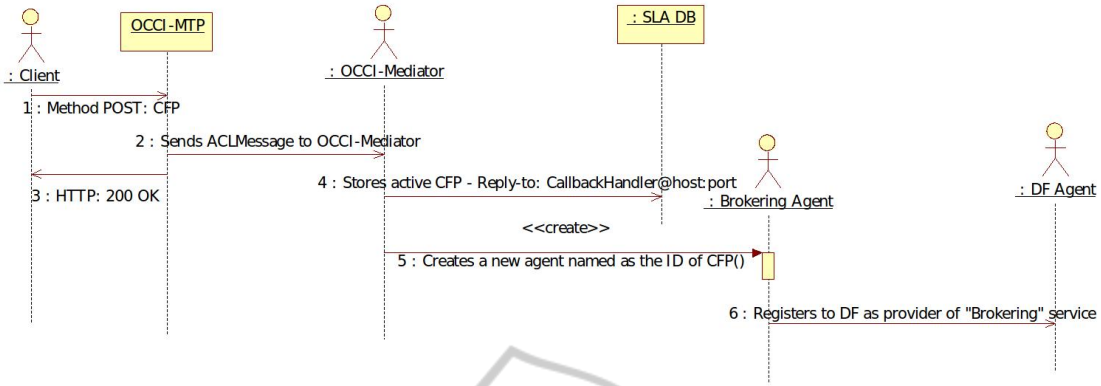


Figure 4: Sequence diagram to start a call for proposal.

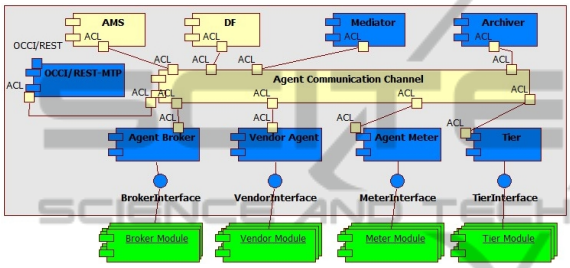


Figure 3: Cloud Agency Components Diagram.

shown in Figure 3. Execution environment for agents and communication facilities are provided by the Jade agent platform(Bellifemine et al., 2003). Jade has been chosen as a platform to provide an execution environment of software agents, an Agent Communication Channel (ACC) and some protocol implementation to support communication. AMS and DF provide standard services of FIPA compliant agent platforms: a name server for agents and a yellow pages registry for publication and discovery of agent base services. Blue components represent agents, which implement the new OCCI MIX-INs. They embed the interaction protocols of the deployment and execution scenarios. Agents will communicate among them via standard ACL (Agent Communication Language) over HTTP, or over other transport protocols if it is necessary. In order to allow a straightforward specialization of agents' behaviors we defined abstract interfaces for brokering, monitoring, reconfiguration and resource management. These abstract interfaces can be implemented by developers who want to extend Cloud Agency by new brokering algorithm, new monitoring technologies, new reconfiguration models or by supporting more Cloud provider. These implementation become plug-ins of CA, which are automatically used by agents through their general interfaces. The OCCI/RESTFull Message Transfer Protocol (MTP) extends the OCCI HTTP rendering (Metsch and Edmonds, 2011).

The call for proposal template includes a list of OCCI resources whose parameters can be initialize with a value, a wildcard or can be associate to a brokering rule.

## 4.3 Cloud Agency: Dynamic View

The Provisioning service of Cloud Agency implements the FIPA Contract-Net protocol(FIPA TC Communication, 2002). It is a minor modification of the original contract net IP pattern in that it adds rejection and confirmation communicative acts. In the contract net IP, one agent (the Initiator) takes the role of manager which wishes to have some task performed by one or more other agents (the Participants) and further wishes to optimize a function that characterizes the task. This characteristic is commonly expressed as the price, in some domain specific way, but could also be soonest time to completion, fair distribution of tasks, etc. For a given task, any number of the Participants may respond with a proposal; the rest must refuse. Negotiations then continue with the Participants that proposed. Cloud Agency uses asynchronous messages RESTfull messages to start a transaction, to notify a Proposal and to agree or refuse. The message encoding is an extension of the OCCI HTTP rendering model described in (Metsch and Edmonds, 2011). The message body is a CFP document. This message triggers the interaction diagram of Figure 4. In Figure 4 the Cloud user prepares a CFP to be sent to the Cloud Agency. The acknowledge of Cloud Agency confirms the correctness of the request and its acceptance for being processed. Cloud Agency contacts all vendors, which provide a bid for that kind of resource. When the available bids have been connected from the Vendor Agents, Cloud Agency returns asynchronously the best proposal that satisfies the CFP requirements. In particular the broker choose the best proposal among the received ones

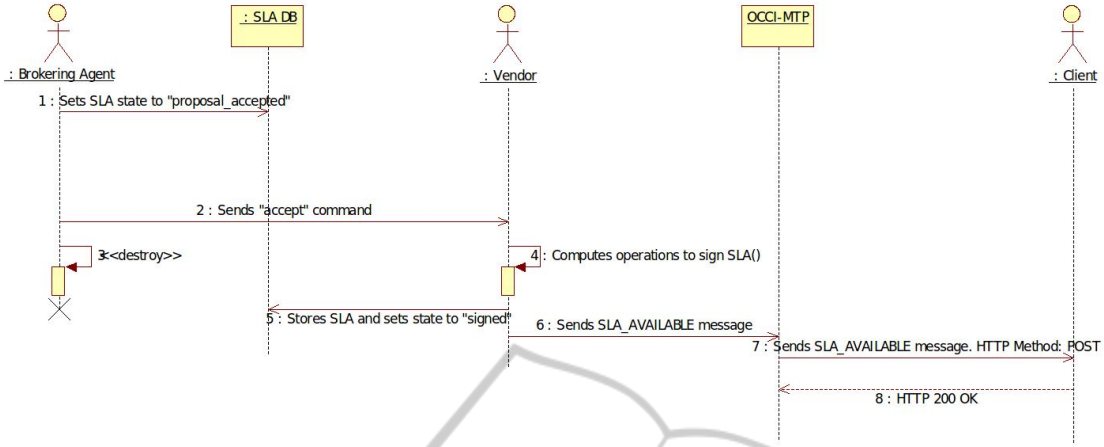


Figure 6: Sequence diagram to notify the availability of a new SLA.

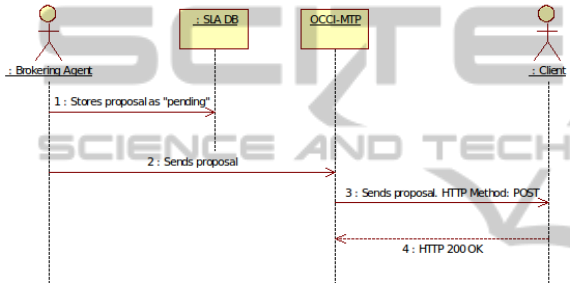


Figure 5: Sequence diagram to notify a new proposal.

and notify the availability of the result to the client 5. When the client has been notified, it can use the location URL from the notification of the available proposal to get the content of the proposal and, by the same message, it can add a query string to the HTTP request to agree or refuse the proposal. The client sends an HTTP GET request to the Cloud Agency. The message is forwarded to the right agent broker that fulfills the request by forwarding the message to the right Vendor Agent. An acceptance or a refuse of the proposal terminates the transaction, and the broker agent itself. In case of agreement the Vendor Agent allocates the related resources and returns a notification about the agreed SLA to the client according to the sequence diagram of Figure 6. After that the resources are available for the application deployment.

# 5 SLA NEGOTIATION: THE mOSAIC APPLICATION

As outlined in section 3, while the cloud agency has the role of managing the brokering, offering it trough a Contract-Net Protocol, implementation of the Negotiation Protocol is delegated to a dedicated cloud application. Such an application has the following main requirements:

- it has an intermediary role between the user and Cloud Agency, implementing the negotiation protocol toward the users.
- it is able to flexibly support many negotiation models.
- it offers a Web interface to its Final Users.
- it is based on SLA negotiation, implementing a complex logic, which is reconfigurable.

We implemented such solution using the mOSAIC framework, which is perfectly integrated with the cloud agency (they were born together), whose details will be described in the following sections, for what is strictly necessary to understand the application.

## 5.1 Flexible Negotiation Models

The cloud application manages a negotiation instance for each user, negotiation steps are modelled trough a finite-state machine (FSM). The solution adopted in order to implement the FSM model in mosaic will be based on already existing components that can be easily adapted to support different kind of FSMs. Changing Negotiation models implies only the needing of sending set of configuration information to already available components. Thanks to the services offered by mOSAIC it is possible to implement such an approach by a little development effort: state information will be stored in cloud-based storage system (this is typical of mOSAIC as it will be shown later), the FSM events will be represented trough messages, while the FSM advances will be defined trough a set of *Policies*. Policy evaluation identifies the advances in the negotiation FSM model. The proposed approach is flexible, in fact it is possible to change the

Negotiation Model, just changing the set of policies stored in the dedicated component.

It would be possible to support a different Negotiation Model for each different user. As already said, our SLA management solution use two finite-state machines: the finite-state machine related to the negotiation model and necessary for the negotiation implementation and the finite-state machine WS-Agreement standard. This model defines the states assumed by an SLA during its management. The WS-Agreement FSM is managed independently by a dedicated mOSAIC component (SLAstore, described later), from now on when we refer to an FSM we refer to negotiation FSM.

## 5.2 mOSAIC Application and mOSAIC Services

In order to illustrate and motivate the way, by which we developed the application, we briefly illustrate what are the features that mOSAIC offers. The interested reader can find detailed description of the mOSAIC Solution in (Rak et al., 2011), (Int, ), (Craciun et al., 2011). In mOSAIC a Cloud application is modeled as a collection of components that are able to communicate each other and that consumes/uses Cloud resources (*i.e.*, resources offered as a service from a Cloud Provider). Components may be offered by the mOSAIC Platform $(i)$ as COTS (Commercial off-the Shelf) solutions, *i.e.*, common technologies embedded in a mOSAIC Component, $(ii)$ as Core Components, *i.e.*, tools which enable application execution and composing the mOSAIC platform, $(iii)$, as Service Components, *i.e.*, tools offered by mOSAIC Platform in order to perform predefined operations , or $(iv)$ as new components that can be developed using mOSAIC API. In last case a component is a *Cloudlet* running in a *Cloudlet Container* (Craciun et al., 2011), (Int, ). mOSAIC Components are interconnected trough communication resources, such as queues. The mOSAIC Platform offers some queuing system (rabbitmq and zeroMQ) as COTS components, in order to enable component communications. Moreover, mOSAIC Platform offers some Service Components in order to help Cloud application to offer their functionalities as a service, like an HTTP gateway, which accepts HTTP requests and forwards them on application queues.

## 5.3 Negotiation Application mOSAIC Components

As anticipated we built up our applications as a mOSAIC Application, which directly communicates with the Cloud Agency. It is a component-based application, in which each component has a different well defined role. Communication between components will take place through messages. In Figure 7 there is the application architecture in terms of interconnected components, which will be described in detail in the following subsection. Each component implements a well-defined functionality:

- `SLAgw` and `SLAstore`, which provides SLA management.
- `TradeProtocolState` and `SLApolicy`, which implements the negotiation model.
- `CAgw`, which interacts with Cloud Agency and assumes decisions on agreement.

In the following we propose a brief description of the behavior of each component, while next subsection tries to outline how they interoperate for the negotiation application. The SLAgw has both the role of offering the application as a Web application and implements a REST-based interface. The SLAgw REST API, accepts essentially the following methods: i) `submit`, with which a Customer submits the agreement or template to Provider, ii) `check`, with which a Customer requests the status of SLA agreement, iii) `sign`, with which a Customer accepts the agreement, iiii) `terminate`, with which a Customer requests the termination of SLA agreement.

Through these methods SLAgw-based applications are able to implement many different negotiation models: the `submit` method will be used to implement many different kind of requests to the underlying application. Moreover `check`, `sign` and `terminate` represent the mandatory operations for SLA management protocols. *SLAstore* is the component that stores an agreement or template with related timeout. It is a component that manages features such as: the capability of taking trace of the signed SLAs and the status of all the exchanged SLAs. In fact, as anticipated, during a SLA management, a SLA can assume several states, defined by WS-Agreement standard. *SLAPolicy* is a component that acts as a Policy Decision Point. It retrieves information, on which performs a policy evaluation and if the evaluation is positive, generates the resulting action, corresponding to the policy. A policy is represented in JSON as a triple of `Parameter`, `Expression`, `Action` lists. The first one contains the parameters to be used for evaluation of the policy, the second one is a logical expression whose variables are the parameters defined before, and action represents the message to be sent if the policy evaluation results true. *TradeProtocolState* has the role of taking track of the states of the negotia-

tion (storing them in the Key Value cloud storage) and acting consequently to the action requests generated by the *SLApolicy* component. During the negotiation process, it communicates with SLApolicy, requiring an evaluation policy operation and with CA which interacts with the Cloud Agency. Moreover it sends out messages to the SLAstorage, both forwarding the new message request sent by the SLAgw and generating new ones in order to update WS-Agreement SLA states. *CAgw* has the role of Cloud Agency Client or gateway: it receives messages from the other components, containing information needed to implement the Contract-Net protocol, then it invokes Cloud Agency APIs.

### 5.4 mOSAIC Application Global Architecture

The application behaviour can be briefly summarized on the base of the messages exchanged by the components which have been described above and connected as in Figure 7. The SLAgw receives the messages from the final users trough its SLA-oriented API. All the negotiation messages are sent trough the *submit* REST call, except the ones needed to effectively sign the SLA. By a web interface the user can interactively send request and check responses. Each message submitted by the SLAgw is intercepted by the *TradeProtocolState* component, which forwards to the SLA storage all the messages containing a new or updated WS-Agreement. Moreover it sends a message to the SLApolicy in order to know how to update the local storage. On the basis of the policy evaluation, which generates *Action* messages, the *TradeProtocolState* updates the local storage with the new negotiation state and, eventually, it sends out messages to the SLAstorage, in order to update the WS-Agreement state and/or sends messages to the *CAgw* in order to starts up requests for the negotiations.

## 6 NEGOTIATION MODEL IMPLEMENTATION

As outlined above, the application is flexible respect to complex negotiation models, but to make it easier to understand, we will focus on a simple proposal-based negotiation model. It is an extension of Contract-Net Protocol that allows the user to choose the best offer by selecting the favourite proposals among a set of available one. It is realized performing many parallel Contract-Net and adding a step for booking the proposal that best fit the user's need, without locking the other choices. The considered negotiation model is shown in 8. It is implemented by several steps. The customer submits a CallForProposal (that consists of several features) to the provider by an agreement template. The provider, once received the template, performs a check and he can reject it or send a template with the supported options . The customer, once received the template, decides to reserve a proposal or to refuse it. If he submits a proposal (that is a SLA), it can be reserved or refused. If a reservation is performed, a time out is set for a SLA agreement. A timeout is a specified period of time that will be allowed to elapse before an customer sign event occurs. If this event takes place before a timeout, the sign operation is performed otherwise not. Such a model is associated to the FSM shown in 9, before offering a detailed description of the FSM implementation few consideration are needed. All the events are labelled with messages, note that messages from users (submit, sign) are explicitly expressed, while the others are labeled as internals (just to do not overload this picture of con-

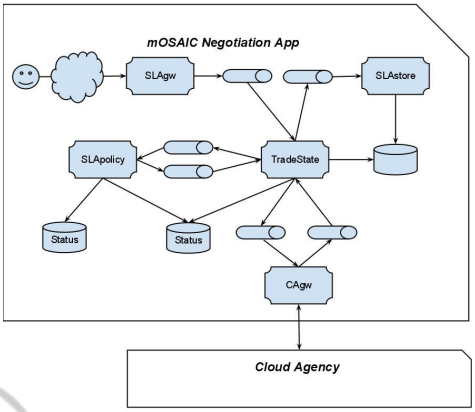


Figure 7: Application architecture.

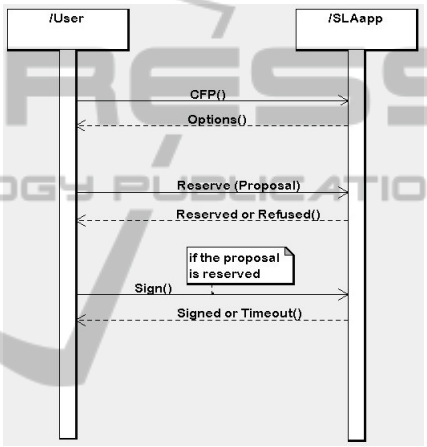


Figure 8: Negotiation Model.

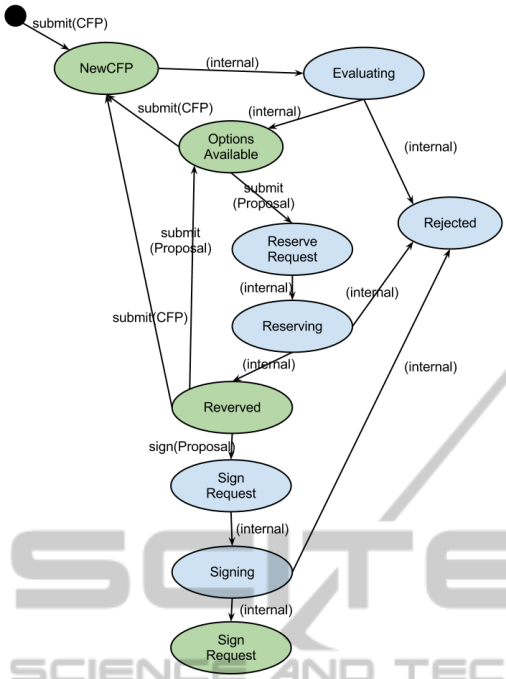


Figure 9: Finite-State machine.

cepts). Internal message names are reported in the table 10, which describe the SLApolicies defined, as actions to be performed. SLA policy evaluation follows the simple rule of getting the State, getting the request and sending out an *action* message which contains the next state and the actions to be performed. Policies are represented in table 10, first column contain the parameters to be retrieved from the KV store (always the same), the second column the expression to be checked, third column the message to be sent if the expression is true. Note that when a policy is true all the others are false. The FSM works as follows: when the final user submits a *template* the first time (the black point) a new ID is created in the KV store by both TradeState and SLAstorage, in order to maintain state information and startup the procedure. The TradeState component, when a submit message of a new WS-Agreement has been received, updates the KV store with a new key (using the Ws-Agreement ID) and a value which contains state information (null at start) and the submitted request, then it sends an evaluation message to SLApolicy. As shown in the FSM and in the policy table, in case of null state SLApolicy emits a message which says that the next state is NewCFP and the action is `CAcfps`. As a consequences TradeState updates the KVstore with NewCFP, forwards the message to SLAstore (that now creates the WS-Agreement state) and prepares the messages to the Cloud Agency. This step strictly depends on the resources to be negotiated: for each possible value of the WS-Agreement template to be independently negotiated, TradeState creates a CFP and sends a request to the CAgw. As a result multiple parallel negotiation will take place trough the Cloud Agency. Note that from now on each tie a new CFP will be submitted by the Final User, it will be discarded, until the FSM arrives to the availability of new options. For each result obtained (*i.e.* a proposal of the Cloud Agency Contract Net protocol) a new message is sent to the TradeState component by CAgw. As shown in the FSM and in the policy table, after the receiving of the first proposal the FSM moves in the OptionsAvailable. TradeState creates a Ws-Agreement template which has as an option the Proposal received. Each new proposal received by the CAgw is added to the WS-Agreement template, enriching it. Note that it is up to the user, that can look at the options available trough the *check* REST call, to wait for other proposals, or cleanup the procedure, submitting a new CFP. In the latter case new proposals will be discarded by TradeState. When the Final User accepts a Proposal, he submits the proposal to the system and Tradestate receives the WS-Agreement an, verifying that it is not a template, but a SLA, moves to the reserve steps. Since now the procedure is completely locked, the only actions allowed are accept or refusal by each of the two parts, as can be seen looking at both FSM and policy list. Note that, in order to have a limited dimension of the policy list, only the main policies are written in the table, while some of them (the most part of the ones that lead to just discard the message) are simply skipped.

| (state,request) | Expression | Action |
|---|---|---|
| (state,request) | (state=null)AND(submitCFP) | (NewCFP,CAcfps) |
| (state,request) | (state=newCFP)AND(CAcfpSent) | (Evaluating,wait) |
| (state,request) | (state=newCFP)AND(submitCFP) | (NewCFP,discard) |
| (state,request) | (state=Evaluating)AND(CAresponse) | (OptionsAvailable,wait) |
| (state,request) | (state=Evaluating)AND(CAtimeout) | (Rejected,end) |
| (state,request) | (state=OptionsAvailable)AND(CAresponse) | (OptionsAvailable,wait) |
| (state,request) | (state=OptionsAvailable)AND(submitCFP) | (NewCFP,CAcfps) |
| (state,request) | (state=OptionsAvailable)AND(submitProposal) | (ReserveRequest,CAreserve) |
| (state,request) | (state=ReserveRequest)AND(CAreserveSent) | (Reserving,wait) |
| (state,request) | (state=Reserving)AND(submitCFP) | (Reserving,discard) |
| (state,request) | (state=Reserving)AND(submitProposal) | (Reserving,discard) |
| (state,request) | (state=Reserving)AND(CAreserve) | (Reserved,wait) |
| (state,request) | (state=Reserved)AND(sign) | (SignRequest,CAsign) |
| (state,request) | (state=SignRequest)AND(submitCFP) | (SignRequest,discard) |
| (state,request) | (state=SignRequest)AND(submitProposal) | (SignRequest,discard) |
| (state,request) | (state=Signing)AND(submitCFP) | (Signing,discard) |
| (state,request) | (state=Signing)AND(submitProposal) | (Signing,discard) |
| (state,request) | (state=Signing)AND(CAsigned) | (Signed,end) |
| (state,request) | (state=Signing)AND(CArefused) | (Rejected,end) |

Figure 10: Negotiation Model Implementation Policies.

## 7 CONCLUSIONS

Support for negotiation of Service level Agreement is a weakness in cloud market nowadays. In fact, at the state of art only few providers are able to offer dynamically configurable SLA services. In this paper, which includes results of the mOSAIC project, we have

shown how it is possible to build a complex negotiation system, which is independent from the cloud provider technologies and allows the user to negotiate the best Cloud service, that is compliant with his requirements. The solution proposed adopts a brokering system, the *Cloud Agency*, in order to acquire autonomically resources from providers on the basis of SLA evaluation rules. Moreover a dedicated application offer to the user negotiation facilities by an interactive interface and according several policies, which can be configured in a flexible way. The application offers a REST-based interface to final users that implements a console for the management of SLAs compliant with the WS-Agreement model.

## ACKNOWLEDGEMENTS

This work has been supported by the FP7-ICT-2009-5-256910 (mOSAIC) EU project and by the MIUR-PRIN 2008 Cloud@Home: a New Enhanced Computing Paradigm.



## REFERENCES

Aknine, S., Pinson, S., and Shakun, M. (2004). An extended multi-agent negotiation protocol. *Autonomous Agents and Multi-Agents Systems*, 8(1):5–45.

Bellifemine, F., Caire, G., Poggi, A., and Rimassa, G. (2003). JADE: A White Paper. *EXP in search of innovation*, 3(3):6–19.

C Emeakaroha, V., Brandic, I., Michael, M., and Dustdar, S. (2010). Low level metrics to high level slas - lom2his framework : Bridging the gap between monitored metrics and sla parameters in cloud environments. *Information Systems Journal*, 2:48–54.

Chhetri, M. and al. (2006). A coordinated architecture for the agent-based service level agreement negotiation of web service compositions. In *The Australian Software Engineering Conference*. Springer.

Craciun, C., Rak, M., and Petcu, D. (2011). Towards a cross platform cloud api. components for cloud federation. In van Sinderen F. Leymann, I. I. M. and Shishkov, B., editors, *Procs. CLOSER 2011 - 1st International Conference on Cloud Computing and Services Science*, pages 166–169. SciTePress – Science and Technology Publications.

Dang, J. and Huhns, M. (2006). Concurrent multiple-issue negotiation for internet-based services. *IEEE Internet Computing*, 10(6):42–49.

FIPA TC Communication (2002). Fipa contract net interaction protocol. Available at http://www.fipa.org.

Metsch, T. and Edmonds, A. (2011). Open cloud computing interface - http rendering. http://ogf.org/documents/GFD.185.pdf.

Nyr n, R., Edmonds, A., Papaspyrou, A., and Metsch, T. (2011). Open cloud computing interface - core. http://ogf.org/documents/GFD.183.pdf.

Rak, M., Venticinque, S., Aversa, R., and Di Martino, B. (2011). User centric service level management in mosaic application. In *Europar 2011 Workshop*. IEEE.

Siebenhaar, M. and al. (2011). Concurrent negotiations in cloud-based systems. In *Proceedings of the 8th International Workshop on Economics of Grids, Clouds, Systems, and Services (GECON2011)*.

Sim, K. and Shi, B. (2010). Concurrent negotiation and coordination for grid resource coallocation. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 40(3):753–766.

Sim, K. M. (2010). Towards complex negotiation for cloud economy. In *5th International Conference on Advances in Grid and Pervasive Computing (GPC 2010)*, pages 395–406.