AGENT BASED CLOUD PROVISIONING AND MANAGEMENT
Design and Prototypal Implementation

Salvatore Venticinque, Rocco Aversa, Beniamino Di Martino
Department of Information Engineering, Second University of Naples, Naples, Italy

Dana Petcu
Institute e-Austria, Timisoara, Romania

Keywords: Cloud, Agents based services, SLA negotiation, Resources provisioning.

Abstract: When interoperability and portability of applications across heterogeneous Clouds are supported, autonomous services which can automatically manage collection, negotiation and monitoring of Cloud resources represents an added value for users within the *sky computing* paradigm. We describe here the design and the prototypal implementation of an Agency that can access the computing utility market relative to the state-of-art of Cloud computing, on behalf of the user, to maintain always the best resources configuration that satisfies the application requirements. This system is in charge to provision the collection of Cloud resources, from different providers, that continuously meets the requirements of user's applications. According to the available offers, it generates a service-level agreement that represents the result of resource negotiation 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.

1 INTRODUCTION

Cloud computing is an emerging paradigm that, due to an intensive use of the virtualization approach, offers resources on which users have full administrative control. A relevant issue affecting this paradigm is related to the mandatory choice of a proprietary solution that the user is asked to take when he wants to Cloudify his applications. The EC-FP7-ICT mOSAIC project (www.mosaic-cloud.eu) intends to improve the state-of-the-art in Cloud computing by creating, promoting and exploiting an open-source Cloud application programming interface and a platform targeted for developing multi-Cloud oriented applications. The main benefit of using the mOSAIC software package will be a transparent and simple access to heterogeneous Cloud computing resources and the avoidance of lock-in proprietary solutions. Once the portability across multiple Cloud is supported many facilities could be provided to improve performability of Cloud applications, or in order to optimize costs and quality of Cloud resource to be bought by switching between different Providers and technological solutions, or using an heterogeneous collection of them according the *sky computing* approach (Keahey et al., 2009). In this context automatic SLA-oriented resource provisioning can represent a solution to negotiate and manage a collection of inter-connected and virtualized computers and storages between resource providers and consumers (or between resource providers and a third-party broker)(Sim, 2010). The most relevant issues to be addressed during this activities are: the definition of an uniform SLA model, that is used to manage internally the ones used by heterogeneous providers; the understanding of users' requirements in terms of Cloud resources which are necessary to run his applications, and are compliant with constraints like costs; the periodic benchmarking model and check of the fulfillment of the SLA by providers; a knowledge model of current and historical information about Cloud; the rules and the actions for automatically detect the bottlencks and apply the solution in terms of reactions. Here we present the design and a first prototypal implementation of an important component of the mOSAIC framework that we called `Cloud Agency` (R. Aversa, 2010) that allows to the user to delegate to the Agency the necessary checks of SLA fulfillment, the monitoring of resource utilization and eventually necessary re-negotiations. Even if security is a mandatory issue

Venticinque S., Aversa R., Di Martino B. and Petcu D..
AGENT BASED CLOUD PROVISIONING AND MANAGEMENT - Design and Prototypal Implementation.
DOI: 10.5220/0003395901840191
In *Proceedings of the 1st International Conference on Cloud Computing and Services Science* (CLOSER-2011), pages 184-191
ISBN: 978-989-8425-52-2

for the authentication of involved parties and certification of SLAs actually it is out of the scope here. The second section presents related work on Cloud resource provisioning and management. In the third Section we discuss the requirements of the Cloud Agency design. The fourth section the architecture of the provisioning subsystem of the mOSAIC platform is described. In the fifth section we provide details about the multi agents model we have designed. Finally prototypal implementation and APIs are described.

2 RELATED WORK

The brokering of Cloud providers whose offers can meet the requirements of a particular application is a complex issue due to the different business models that are associated with such computing systems. According to (R. Buyya and Brandic, 2009) a market-oriented resource management is needed in order to regulate the supply and demand of Cloud resources, providing feedback in terms of economic incentives for both Cloud consumers and providers, and promoting QoS-based resource allocation mechanisms that differentiate service requests based on their utility. The current Cloud computing technologies offer a limited support for dynamic negotiation of SLAs between participants. There is no mechanisms for automatic allocation of resources to multiple competing requests. Furthermore, current Cloud computing technologies are not able to support customer-driven service management based on customer profiles and requested service requirements. It is impossible, according to (R. Buyya and Brandic, 2009), to derive appropriate market-based resource management strategies that encompass both customer-driven service management and computational risk management to sustain SLA-oriented resource allocation. An attempt to define several QoS metrics is presented in (B. Cao and Xiang, 2009): response time, availability, reliability, cost and reputation are considered. A reference of SLA model is provided in (Sim, 2010) where SLA objectives (SLOs) are used to compose an SLA. A number of service levels and performance metrics for each resource results in multiple SLOs for every service. The work presented in (A. Kertesz, 2009) represents a first proposal to combine SLA-based resource negotiations with virtualized resources in terms of on-demand service provision. The architecture description focuses on three topics: agreement negotiation, service brokering and deployment using virtualization. It involves multiple brokers. A Cloud multi-agent management architecture is proposed in (Cao et al., 2009). It includes requestor agents, QoS agent, provider agent and agent manager. In particular the QoS agent is dedicated to the submission of QoS information about services. A simpler agents based architecture has been proposed in (X. You and Yu, 2009). It is simpler than the one described in this paper and consists of only three parts: consumer agent delegated by the consumer to obtain a maximal benefit for him, resource agent delegated by the resource provider to publish the prices and to adjust them according to the relationship of supply and demand in the market system and the market economy mechanism responsible for balance the resource supply and the demand. New SLA-oriented resource management strategies must be designed for Clouds in order to provide personalized attention to customers. Service requirements of users can change over time, due to continuing changes in business operations and operating environment, and thus may require amendments of original service requests. Effects due to concurrent negotiation activities between broker and provider agents in multiple Cloud resource markets have to be considered. The negotiation that outcome between broker and provider agents influence the negotiation outcomes of broker and consumer agents in a Cloud service market. The implementation that was presented in (Sim, 2010) is very interesting from this point of view because it is supporting dynamic negotiations.

3 REQUIREMENTS

SLA Negotiation with multiple Cloud providers is a first example of complex application that could be delegated to a third party, represented by a broker in a market based context. A broker intermediates between users and providers in order to negotiate the best SLA for both consumer and vendors. On user behalf it can search for available Cloud services, which are compliant with user needs; check of trustiness of providers; decide with whom to negotiate, according to user requirements and past experiences; negotiate the best price for the same offer by different providers; negotiate multiple SLAs, with different providers, to overcome the lack of one compliant offer by a single provider. Since consumers' requirements can potentially vary over time it needs to support dynamic re-negotiation of SLA. Some mechanisms to reconfigure virtual resources are already available, but it needs policies and protocols for changing the SLA parameters, to include new amendments and withdraw previous ones. Re-negotiation is another service that can be provided to solve some inconsisten-

cies between the SLA and the real user's requirements which can change dynamically. Dynamic SLA re-negotiation has actually limited support. Issues to be investigated are: withdraw of an SLA and negotiation of a second one; deletion of an SLA objective; addition of an SLA objective; redefinition of SLA parameter; negotiation of boundaries within which the SLA can be re-negotiated at the same price or with a pre-defined price adjustment.

Monitoring the utilization of Cloud resources is another service that can be delegated. Providers monitor utilization of their resources for billing, to change bid prices in order to optimize profit, to not exceed in resource allocation beyond the capability of fulfill the agreements. On the other hand, the user, who has conflicting interests with providers, needs to trust a third party that can be delegated to monitor the satisfaction of the agreed service levels. Monitoring process should provide information about under-utilization of cloud resources, in order to negotiate cheaper agreements; saturation of resources, to not let the users' applications work under the QoS level granted to users' clients; unbalanced utilization of Cloud resources, in order to check the correctness of negotiated parameters, or to tune the execution of applications in the Cloud; violation of SLA by providers.

4 MOSAIC PROVISIONING SYSTEM

The Provisioning services of the mOSAIC framework are in charge to implement a set of functionalities for resources negotiation and management. These facilities can be used by the mOSAIC Runtime Engine, if the developer uses the mOSAIC SDK to implement his applications and wants to delegate the resource management. Otherwise these services can be invoked independently by the user application, if it has the knowledge itself about how to use the provisioning facilities in a custom way. Two set of functionalities ar provided: methods which can be used to get information about the booked Cloud resources and services to build a specific workflow for resource (re)negotiation. The identified actions will be available to the other components of the mOSAIC framework and to the user application by two packages of APIs. A component diagram representing the Provisioning sub-system of the mOSAIC framework and its connection with the other components is showed in Figure 1. *The CloudAgency* is a multi agent system that is responsible for SLA management: negotiation, re-negotiation, monitoring. *The Negotiation*

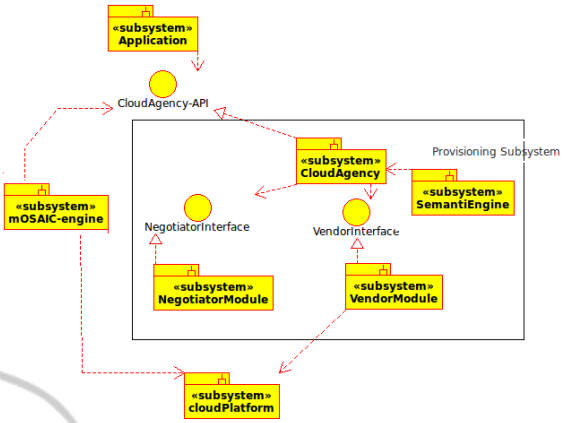

Figure 1: Components diagram of the Provisioning Subsystem.

Module that implements different policies for composing the more effective SLA according to the application requirements and the Vendors' proposals. *The Vendor Module* which can be implemented by different technologies to support the interaction with Cloud Platforms to acquire and to instantiate resources. *The Semantic Engine* that is able to discover in the Network new Cloud Services and Providers and to update the list of available ones if they are supported by mOSAIC. The Negotiator Interface and the Vendor Interface are abstraction of the Negotiation Module and the Vendor Module. They allow mOSAIC developers to implements different policies, without being aware about the agents technology and about the Cloud Agency implementation.

4.1 Semantic Engine

One of the most challenging goals of the semantic engine is to design and develop semantic-based Cloud services discovery. This component searches in the Network for Semantic description of Cloud services and resources. WSDL, WSDLS-S, semantic annotations are processed in order to discover new available providers which are supported by mOSAIC. The Semantic Engine uses the mOSAIC Cloud Ontology and a reasoning module to identify he resource/service type, the Cloud platform interface and the bindings to the provider. An example can be represented by a new Cloud Provider that offers a storage that can be used by an Amazon S3 compliant interface. In this case the new available service is published in the DF (Directory Facilitator) and will be taken in consideration by the Mediator Agent during the brokering. Reasoning will be based on syntactic and structural schema matching. The input will be an ontology describing the Cloud knowledge that include

resources, techniques, technologies and services descriptions. This can be achieved on the syntactic level through a service description language (like WSDL), or on the semantic level, through service ontologies (like in OWL-S and WSDL-S). Semantic matching is possible since services descriptions are semantically annotated based on concepts from ontologies adopted for modeling the specific domain of application. The result of a semantic discovery is a new set of exploitable Cloud services and providers. *Semanti Engine* and *Cloud Ontology* are two specific outcomes of the mOSAIC project and are not further addressed here.

4.2 Cloud Agency

As it is shown in Figure 2 the Cloud Agency architecture is composed of different components. The *Agent Platform* provides the environment for executing agents and the basic facilities to manage the agent life cycle. It also provides communication facilities which are used to implement agents interaction protocols. A group of agents interact to provide the mOSAIC services which can be used by a set of APIs. The **DF** is an agent whose presence is mandatory in every agents platform designed according to the FIPA(A. Grama and Sameh, 2000) standard. It provides yellow pages. Here it is used to publish the available services and resource provided by the Vendor Agents. The **AMS** is an agent that manages agent names and agent addresses. Also the AMS is a mandatory component of FIPA standard agents platforms. mOSAIC Agents are new components of the agency which have been designed according the framework requirements discussed above. A **Client Agent** is in charge to receive request from the user, from the application or from the mOSAIC runtime. It acts as access point for the Cloud Agency services.

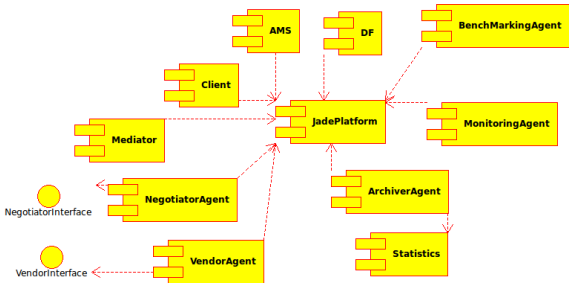

Figure 2: Component diagram of the Cloud Agency.

It receives the user requests and cooperates with the Negotiator in order to broker Cloud services and resources with the requested quality levels. The **Negotiator Agent** receives from the Client an SLA Template (a call for proposal) and splits the SLA Template in different objectives which will be forwarded by the Mediator Agent. It receives different proposals from one or more Vendors. The Negotiator Agent uses the Negotiator interface to parse the SLA Template, to evaluate the proposals and to compose the final SLA to be returned to the Client Agent. The **Mediator** is a broker that, for each call for proposal coming from the Negotiator Agents, searches in the DF for the available vendors which can answer to that call. It contacts each Provider Agent and requests a bid for the needed resources. Once it obtains responses from Provider Agents, it assesses the following: the QoS provided; the quality of the provider itself (requesting historical data from an Archiver Agent). After he assessed the bid responses, they are forwarded to the the Negotiatior that puts together a contract with the winning providers on behalf of the client. The Negotiator Agent could try to optimize the contract by applying different trade-offs among performance, availability or costs, but within the bounds specified by the client. The **Archiver Agent** maintains statistics about the mOSAIC system. It use Benchmarking Agents to measure QOS parameters and to store their values. It also can be used by external components, such as the mOSAIC runtime, or the applications, to store and to load statistics. **Benchmarking Agents** collect measures about the QOS parameters to be monitored. They can perform measures executing into the Cloud, within each virtual machine. The **Monitoring Agent** is responsible to use the statistics in order to check the compliance of the current performances with the Service Level defined in the SLA. It can also inform users' own applications or renegotiation services about the real utilization of the acquired resources in order to detect critical working condition (e.g.: saturation or underutilization). The **Negotiator Interface** will be implemented by the Negotiator Module, that represents the brain of the Negotiator Agent. It is used to update the received proposal and combine them in order to generate the final SLA. The **Vendor Agent** talks on one side with the Mediator and on the other side with the Cloud Provider. Such as a wrapper, or a driver, it translates the specific provider protocol into the uniform protocol used inside the mOSAIC framework. It accepts bid requests from the Mediator and try to propose a contract for the resources it can provide. The Vendor Interface will be implemented outside the Cloud Agency to support multiple Cloud technologies. The Vendor Interface will be implemented by the *Vendor Module* that is aware about a specific Cloud technology and is able to interact with a particular provider in order to ask for proposals, accept them and instantiate

resources getting the bindings to be used by applications or by the mOSAIC runtime.

5 AGENTS' BEHAVIOURS

The cooperation among agents has been designed adopting well defined Agents Interaction Protocols of the FIPA standard. They have been combined as follows. The Client Agent implements a standard FIPA Contract Net Interaction Protocol with the Negotiator Agent. It submit a call for proposal using an SLA Template that describes the application requirements and gets the SLA proposal for acceptance or refusal. The Mediator implements the standard FIPA Brokering Interaction protocol. The sub protocol used to talk with vendors is again the standard FIPA Contract Net interaction protocol. The Vendor Agent implements the Standard FIPA Contract Net interaction protocol at Provider Side.

5.1 ClientAgent

The automata representing the behavior of the Client Agent is shown in the Figure 3.

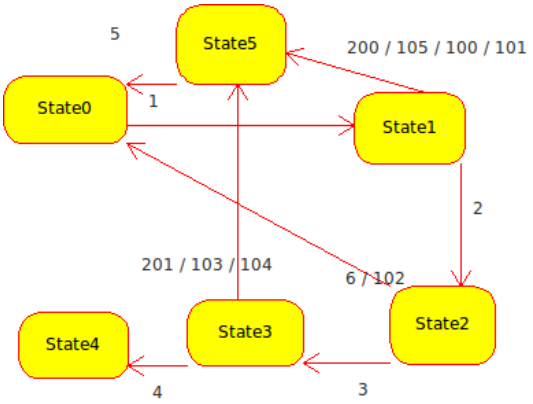

Figure 3: Client Agent.

- State0: it's the first state of our Client Agent. In this state the agent is ready to receive application requirements. The event 1 corresponds to the reception of requirements and on its occurrence the client send the Call For Proposal message to the Negotiator Agent.
- State1: in this state the agent waits for a message from the Negotiator and if it's a propose message the client gets the SLA Proposal, it's represented by event 2. Other events are:
  1. 101 → the client receives a REFUSE message
  2. 100 → the client receives a FAILURE message
  3. 105 → the SLA in the message is unreadable
  4. 200 → the protocol is incorrect
- State2: the client waits for the user to accept, refuse or renegotiate the SLA. If the user agrees the SLA is signed and sent to the Negotiator, event 3 is generated. Else if the user refuses the SLA proposal, event 102 is generated. Finally if the user wants to renegotiate the SLA event 6 is generated.
- State3: the client waits for a reply message from the Negotiator. If the message is an inform message than the client updates the SLA (it's within the message) and launches event 4. If it receives a FAILURE the message event 103 is launched, unreadable content is associated to the event 104 and unknown protocol/message is associated to the event 201.
- State4: it's the final state, it informs that the resources are available and starts the Monitor Agent.
- State5: it shows the failure/refuse or unknown messages received from the Negotiator. This state launches the event 5.

5.2 Mediator Agent

The automata representing the behavior of the Mediator Agent is shown in Figure 4.

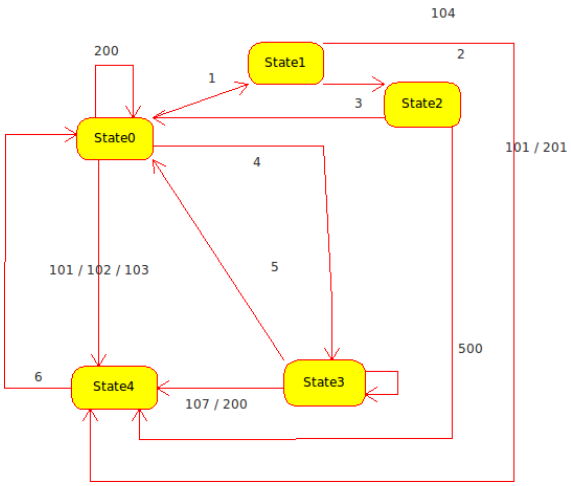

Figure 4: Mediator Agent.

- State0: the Mediator waits for a proxy or a subscribe message. When it receives a proxy message it searches for the available vendor agents who can provide the required resource/service and launches event 3. Otherwise if it does not find any

vendors it throws event 101. If it receives a message with unreadable content event 103 is generated. If the Mediator receives a subscribe message it sends the SLA that is within the message to the vendor by an *accept_proposal* message. Event 200 means an unknown message/protocol.

- State1: the mediator sends the objective to the Vendor (event 2 is generated). If it cannot insert the objective into the message, it throws the event 104.
- State2: in this state the Mediator waits for the vendor's reply. If it receives a propose message the Mediator sends an *inform_done_proxy* message to the Negotiator Agent and then it sends the *reply_message_sub_protocol* message to the Negotiator. To this last message the Mediator attaches the Vendor's bid and launches the event 3. If it receives a refuse message, it throws an event 101. If it receives an unknown message/protocol it generates an event 201.
- State3: the Mediator waits for the vendor's reply, if it is an *inform_done* (this message contains the SLA signed by vendor) the mediator sends an *agree* message with SLA attached to the Negotiator. The associated event is event 5. If the vendor receives a failure message it generates event 107 or event 200 if the message is an unknown message/protocol.
- State4: in this state the Mediator manages the unknown message/protocol, failure message, refuse message and events 103, event 104 (events associated with Input/Output error on message content).

5.3 Negotiator Agent

The automata representing the behavior of the Negotiator Agent is shown in Figure 5.

- State0: the Negotiator waits for the CFP message and extracts from the SLATemplate the objectives list. Then it starts the event 1. Other events are:
  1. 200 → unknown message/protocol
  2. 101 → unreadable content of message
- State1: if all objectives are processed event 6 is launched, else the Negotiator sends a proxy message to the Mediator and launches the event 2. Event 101 represents unreadable objective.
- State2: in this state the Negotiator waits for the Mediator's reply. If the reply is an agree message the Negotiator launches the event 3, else if the message is a refuse message we have event 100. If it receives an unknown message/protocol we have event 200.

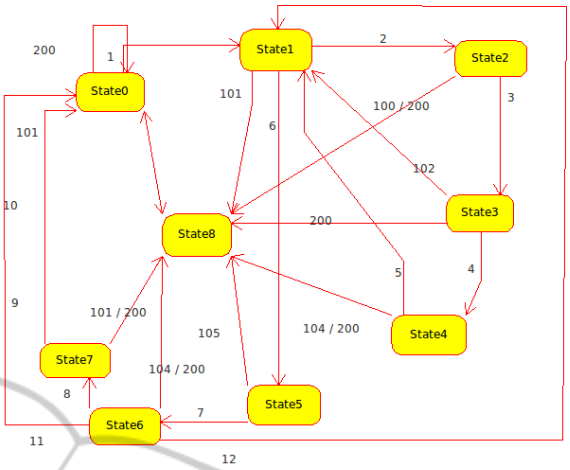

Figure 5: Negotiator Agent.

- State3: the Negotiator waits for a message from the Mediator. The association between event and message are:
  1. 102 → *failure_not_match*
  2. 4 → *inform_done_proxy*
  3. 200 → *unknownmessage/protocol*
- State4: the Negotiator waits for the *reply_message_sub_protocol* from the Negotiator. In this way it can update the archive. In this case event 5 is launched. If the objective within the message is unreadable it launches event 104, if the message is unknown or out of protocol, event 200 is launched.
- State5: this state produces the SLA and sends it to the client by a propose message (event 7), if the Negotiator cannot set the content of the message it launches an event 105.
- State6: the Negotiator waits for the client's reply. If the reply is a cfp the Negotiator updates the objectives list and launches event 12. If the content of the cfp message is unreadable it generates event 105. When the negotiator receives an *accept_proposal* message (the content of this message is the SLA signed by the client) it sends the SLA to the Mediator by a subscribe message, event 8 is generated. Event 200 represents an unknown message/protocol.
- State7: the agent waits for the mediator's reply. If it receives an agree message (it contains the SLA signed by client and vendor) it sends the SLA to the client by an inform message, event 9 is launched. When this does not happen and it receives a failure message, event 101 is generated. Event 200 represents an unknown message/protocol.

- State8: this state manages the unknown message/protocol, failure, refuse message. In this state a message is sent to the Client and to the Mediator to restore their State0. Event 10 is launched.

6 PROTOTYPAL IMPLEMENTATION

A prototypal implementation of the Cloud Agency has been developed using the JADE technology. JADE is a FIPA compliant agent framework that supports development and execution of agent based applications. DF and AMS are provided by JADE as mandatory components of the agent platform. Agents can execute in a single instance of the Jade platform or can be distributed on multiple machines. The Cloud Agency can execute on the user machine or in the Cloud. According to the model described above agents exchange message to communicate among them. Persistent information about resources, monitoring, transactions are stored in a database accessible by a common interface. The services provided by the agency can be used by a request-inform protocol. An ACL message over http must be sent to the Client to ask for a services and to get the response. A first implementation of the mOSAIC APIs for provisioning has been provided in Java, but any language could be used. At the bootstrap each available implementation of Vendor Agent is created. The Vendor publishes in the DF the services that he can provide. On the other hand the client is able to look for all supported resources that can be acquired. Actually we have a Vendor for the Perf-Cloud platform (E. P. Mancini and Villano, 2009) that is able to provide virtual machine as Cloud resource. Perf-Cloud is our own mOSAIC Cloud environment. It is under construction and intends to offer common capabilities of the current IaaSs. We are going to develop agent vendors for other kinds of commercial and open platforms and for heterogeneous cloud resources like storages. If the user wants to use services of our prototype interactively, we provide a GUI that allows to specify the values for each relevant parameter of the resource to be acquired. Actually the user can define the configuration of one or more virtual machines (memory, number of CPU, connection speed) and submit the request. In Figure 6 the Jade Sniffer shows the message exchange among Jade implementation of Client, Mediator, Negotiator and Vendors. User's request is translated into an SLA template that follows actually a simple model defined for experimental purpose. The SLA template is sent to the Negotiator Agent to begin

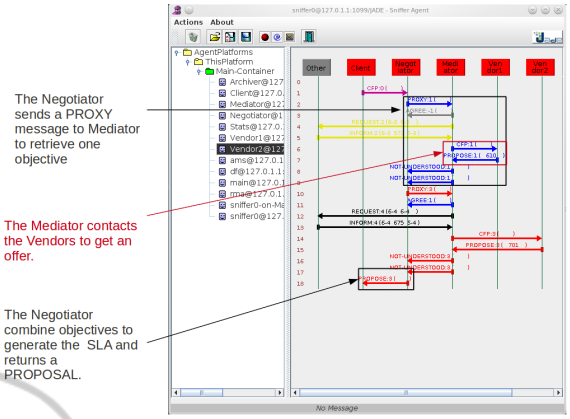

Figure 6: Implementation of the interaction protocol in Jade.

a new transaction. The transaction is univocally identified by a code that is univocally and automaticaly generated a this step. The code is used as conversation ID in all communications among agents which collaborate to complete that transaction. The Negotiator ask for an offer to the Mediator that satisfies the requirements of the single objective (a virtual machine) requested by the user. In the current implementation the Mediator chooses one provider among the available ones and forward the request. The received offer is stored into the database and is returned to the Negotiator Agent. The Negotiator module is responsible combine offers in order to verify if all requirements are satisfied and to build the best contract for the client. Actually the negotiation algorithm is very simple. The negotiator module checks that all values of the parameters in the SLA template are less equal than the ones of the offer. When this does not happen the contract is built too, but a FAILURE is notified. Additional information is attached to specify the reasons of the failure. The user can refuse or accept the contract, also in a FAILURE status if the failures are considered not relevant. In Fig. 7 it is shown the graphical user interface that allows the user to view the SLA, to get the status and to accept or refuse it. When the SLA is accepted the negotiator is notified and he ask to the mediator to confirm the offer and the allocate all the acquired resources. At this moment the agent vendor in the described example asks to the Perf-Cloud provider to allocate the VM and to start it. When the VM is started the Perf-Cloud provider returns the binding to the resource in the form of a ssh URL and credentials. The binding is attached to the SLA and the contract turns into a READY status. A reconfiguration is supported according to a simple withdraw-negotiate model. We mean that if users need a more powerful machine they must negotiate a new SLA. After that they must ask

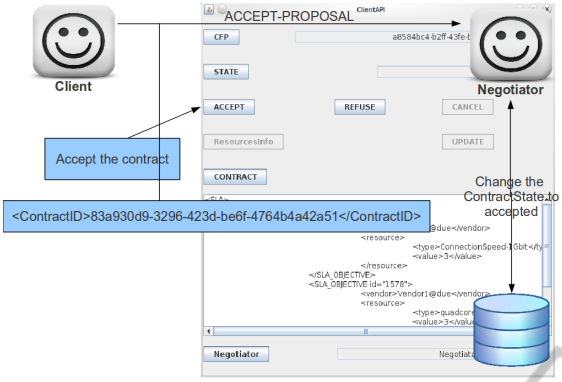

Figure 7: Client Agent.

for a withdrawal of the old one. The agency allows to the users to withdraw a single objective of a composite SLA if the provider supports this action.

7 CONCLUSIONS

In this paper we presented the design and the prototypal implementation of agent based services for Cloud resource provisioning. It is a component of a framework that aims at supporting development of portable multi-Cloud applications. A set of synchronous APIs have been designed to be used within a Cloud Application or by an interactive graphical user interface in order to ask for heterogeneous Cloud resources to multiple providers. APIs can be implemented as XML requests over HTTP in order to be independent from the chosen technology and to support the utilization of different programming languages. The current solution based on the Jade technology is already available. At the state the art negotiation is the first working service of the agency. Monitoring of resource utilization, checking of fulfillment of the SLA, re-negotiation will be next services to be designed and integrated. Furthermore a number of commercial Cloud providers and open Cloud platforms will be supported. Finally an expert system will allow semantic discovery of new Cloud services and resources in order to increase the market within which the negotiator can look for, exploiting also intelligent services composition.

ACKNOWLEDGEMENTS

This research is supported by the grant FP7-ICT-2009-5-256910 (mOSAIC).

REFERENCES

A. Grama, V. K. and Sameh, A. (2000). Foundation for intelligent physical agents.

A. Kertesz, G. Kecskemeti, I. B. (2009). An sla-based resource virtualization approach for on-demand service provision. In *VTDC09 - The 3rd International Workshop on Virtualization Technologies in Distributed Computing*.

B. Cao, B. L. and Xiang, Q. (2009). *A Service-Oriented Qos-Assured and Multi-Agent Cloud Computing Architecture*, volume LNCS 5931 of *CloudCom 2009, High Performance Computing: Paradigm and Infrastructure*, pages 644–649. Springer-Verlag Berlin Heidelberg.

Cao, B.-Q., Li, B., and Xia, Q.-M. (2009). A service-oriented qos-assured and multi-agent cloud computing architecture. In Jaatun, M. G., Zhao, G., and Rong, C., editors, *CloudCom*, volume 5931 of *Lecture Notes in Computer Science*, pages 644–649. Springer.

E. P. Mancini, M. R. and Villano, U. (2009). Perfcloud: Grid services for performance-oriented development of cloud computing applications. In *Proc. of Emerging Technologies for Next generation GRID (ETNGRID-2009/WETICE-2009)*.

Keahey, K., Tsugawa, M., Matsunaga, A., and Fortes, J. (2009). Sky computing. *IEEE Internet Computing*, 13:43–51.

R. Aversa, B. Di Martino, M. R. S. V. (2010). Cloud agency: A mobile agent based cloud system. In *Procs. 2010 International Conference on Complex, Intelligent and Software Intensive Systems*, pages 132–137.

R. Buyya, C. S. Yeo, S. V. J. B. and Brandic, I. (2009). Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. 15(6):599–616.

Sim, K. (2010). Towards complex negotiation for cloud economy. In Bellavista, P., Chang, R.-S., Chao, H.-C., Lin, S.-F., and Sloot, P., editors, *Advances in Grid and Pervasive Computing*, volume 6104 of *Lecture Notes in Computer Science*, pages 395–406. Springer Berlin / Heidelberg.

X. You, X. Xu, J. W. and Yu, D. (2009). *RAS-M: Resource Allocation Strategy Based on Market Mechanism in Cloud Computing*. 2009 Fourth ChinaGrid Annual Conference.