Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 68 (2015) 3 – 16

Procedia Computer Science

HOLACONF - Cloud Forward: From Distributed to Complete Computing

Smart Cloud Engine and Solution based on Knowledge Base

Pierfrancesco Bellini, Daniele Cenni, Paolo Nesi *

Distributed Systems and Internet Technology Lab, DISIT lab, http://www.disit.dinfo.unifi.it

Department of Information Engineering http://www.dinfo.unifi.it University of Florence, http://www.unifi.it, Florence,Italy

Email: {pierfrancesco.bellini, daniele.cenni, paolo.nesi}@unifi.it

Abstract

Complexity of cloud infrastructures needs models and tools for process management, configuration, scaling, elastic computing and healthiness control. This paper presents a Smart Cloud solution based on a Knowledge Base, KB, with the aim of modeling cloud resources, Service Level Agreements and their evolution, and enabling the reasoning on structures by implementing strategies of efficient smart cloud management and intelligence. The solution proposed provides formal verification tools and intelligence for cloud control. It can be easily integrated with any cloud configuration manager, cloud orchestrator, and monitoring tool, since the connections with these tools are performed by using REST calls and XML files. It has been validated in the large ICARO Cloud project with a national cloud service provider.

Keywords: knwoledge base, smart cloud, cloud computing, service level agreement

1. Introduction and related work

Any relevant software infrastructure is presently deployed on cloud to manage resources in an efficient manner. The resource management is becoming a relevant and hot topic for these solutions that need to provide high availability and quality of service. Therefore, specific solutions for cloud monitoring, analyzing and changing configurations and services in the cloud are becoming mandatory to increase resilience and reliability. To this end, the modeling and formalization of cloud resources and information are becoming more relevant to manage different aspects of a cloud at its different levels: IaaS, PaaS, SaaS, and towards specific resources: hosts, virtual machines (VM), networks, memory, storage, processes, services, applications, etc., and their relationships. Cloud

* Corresponding author. Tel.: +39-055-2758515; fax: +39-055-2758570.
E-mail address: paolo.nesi@unifi.it

doi:10.1016/j.procs.2015.09.219

infrastructures are becoming every year more complex to be managed especially for process configuration and reconfiguration, dynamic scaling for elastic computing, resources healthiness control, etc. Several thousands of different resource kinds are available and corresponding relationships among the specific instances of them on the cloud can be established. Thus, every day new resource models and types are added on the market, increasing complexity and demanding very high level of flexibility in cloud management and optimization. The resources are obviously related to elements on cloud such as: hosts, VM, services, storages, processes, software applications, networks, etc., and thus on their corresponding composition and Service Level Agreements, SLA. The SLA can be regarded as the contract associated with a set of cloud resources and their configuration contributing to a given service. For example, a SLA is signed by the Customer with a Cloud Service Provider, CSP; and the SLA describes its requested and requirements with respect to cloud services configuration and quality (e.g., 5 hosts, 4 CPUs at 5000 MHz, 3 VMs in a certain configuration, network minimum of 50 Mbps). And thus, with the SLA the CSP declares the service level guaranteed and the constraints associated with the usage of cloud services. To this end, a number of metrics are assessed for computing the business costs of the business on cloud in "as a service" basis, and are used to verify the SLA conformity. A review about SLAs models and kinds offered by commercial Cloud Providers can be obtained from Wu and Buyya, 2011[1].

As a general consideration, the automation of cloud management may imply to cope with a number of activities also in respect of the SLA, such as: (i) formal verification and validation of cloud configuration in terms of resources, their relationships and matches (this action, can be performed before or after the resource deploy; when it is performed before it can be regarded as a sort of simulation with respect to the available resources); (ii) verification and reasoning about cloud security taking into account networking and storage aspects; (iii) facilitating interoperability among public and private clouds, and/or among different cloud segments managed by different cloud orchestrators or managers in the same infrastructure; (iv) discovering and brokering services and resources; (v) reasoning about cloud workload conditions, may be via simulation; (vi) computing capability for horizontal and/or vertical scaling, thus elastic computing. In the literature, these aspects are addressed in several different manners. Some of them into SLA brokers such as in Cuomo et al., 2012[2], and by Pengcheng et al., 2011[3], while for the minimal monitoring you can see Ward and Barker, 2014[4].

Therefore, reasoning tools about constraints and configurations are needed, and they have to model the infrastructure, the resources and the rules to manage them with respect to the real data collected on the field.

In a seminal work of Youseff et al., 2008[5], an approach to create a cloud ontology has been proposed decomposing cloud modeling problems into five layers: applications, software environments, software infrastructure, software kernel, and hardware. Moreover, Zhang et al., 2012[6] proposed a solution to make easier the search services and resources into the cloud by presenting the CoCoOn ontology, also integrating other QoSOnt ontology (Dobson et al., 2005[7]) for the description of service discovering and parameters. For the description at level of IaaS, the INDL ontology (Infrastructure and Network Description Language) defines nodes connected via links and interfaces in Ghijsen et al., 2012[8]. Virtual Nodes are used to model VMs in execution on former nodes. Node Components are used to represent resources as memory, storage, CPU, while many details are missing as the real network address and also many other physical aspects of VM towards Hosts. In mOSAIC EC project (Moscato et al., 2011[9]), the cloud knowledge modeling has been addressed with the aim of creating a common model to cope with the heterogeneity of different clouds vendors, and with systems with different terminologies. mOSAIC ontology allows to describe aspects of hosts and VM, while presents some lacks on the modeling connections. For modeling the main entities of cloud some general ontologies could be used. For example, https://geni-orca.renci.org/owl/owl-test/compute.rdf# ontology has been defined into ORCA project https://geni-orca.renci.org, that is mainly focused on modeling the hierarchical aspects of networking, and limited capabilities in modeling cloud entities and applications on the whole cloud levels of NIST.

For the description of general cloud services, Linked-USDL in Pedrinaci et al., 2014[10], it provides a set of ontological model for the description of services, as SLA, security, price and intellectual property. Linked USDL (Unified Service Description Language) is a remodeling of USDL language and reused other RDF(S) vocabularies such as GoodRelations (http://www.heppnetz.de/projects/goodrelations/), Minimal Service Model (http://iserve.kmi.open.ac.uk/wiki/index.php/IServe_vocabulary), FOAF (http://www.foaf-project.org/). At application level of the cloud, the standard OASIS Topology and Orchestration Specification for Cloud

Applications (TOSCA), of Binz et al., 2012[11], and in Binz et al., 2014[12], allows to describe via XML the application components, their dependencies, the plan (workflow) for provisioning on the infrastructure, thus formalizing the work of the Orchestrators. In the work of Bernstein et al., 2010[13], the ontological model is used for describing intercloud architectures adopting an ontology for describing cloud services. This approach is at the basis of the IEEE P2302 standard (https://standards.ieee.org/develop/project/2302.html) and exploits the mOSAIC model for the services and resources with the related limitations. Currently, there are a few efforts in building smart cloud solutions grounded on ontology on cloud computing, Androcec et al., 2012[14].

It should be considered as more lightly related work, the specific modeling of services provided the OWL-S has been proposed for describing web services with the service profile, and the WSDL. For the SLA of Web Services, WSLA has been proposed via an XML schema, and allow composing metrics on specific services, Ludwig et al. 2003[15]. In this context, WS-Agreement has been developed by "Grid Resource Allocation Agreement Protocol Working Group" (GRAAP-WG) with the aim of describing SLA among distributed entities in the grid (Andrieux et al. 2007[16]). On the basis of the WS-Agreement, in Oldham et al., 2006[17], an ontology has been defined. All these results have not been open to the public and are strongly focused on web services and related quality of service.

The work presented in this paper reports a Smart Cloud solution based on modeling cloud resources and information via a Knowledge Base, KB. The proposed KB also includes SLA and detailed descriptions and monitoring information associated with resources, thus allowing the monitoring of the SLA evolution, managing complex configurations, related SLA and related strategies for dynamic scaling and elastic computing. In this context, the SLA is modeled and addressed as the service agreement from the CSP and the cloud customer and not at level of web service, or network service. The adoption of a KB to model the cloud knowledge grounded on a cloud ontology and data instances enables the reasoning on cloud structures and their evolution. And thus, it is suitable for implementing strategies of smart cloud management and intelligence. Moreover, the proposed Smart Cloud solution can be easily exploited in connection with any cloud management tools such as configurators, orchestrators, and monitoring tools. The proposed solution has been developed in ICARO Cloud project and validated on the cloud infrastructure of Computer Gross. Computer Gross is a CSP, providing cloud services at different levels: IaaS, PaaS and SaaS, in which allocated applications as SaaS level are provided by several different vendors and belong to categories of multitier solutions for CRM (Customer Relationship Management), ERP (Enterprise Resource Planner), workflow, marketing, business intelligence, etc. This variety of solutions deployed on cloud increases complexity in the cloud management, and motivates the need of flexible smart cloud engine as ICARO. The validation phase of the proposed Smart Cloud solution has been focused on assessing its effectiveness with respect to the automation of scaling, dynamic scaling, reconfiguration, and continuous verification of SLA and resource healthiness.

This paper is structured as follows. In Section 2, the ICARO Smart Cloud architecture is presented in relationships with the typical elements on the different layers of cloud infrastructures. Section 3 describes the ICARO cloud ontology for modeling the cloud at the basis of the Knowledge Base adopted for smart cloud reasoning about configurations, status conditions, scaling, and SLA. In Section 4, the structure and solution for the Smart Cloud Engine are presented. Section 5 depicts some experimental results that can give you the effectiveness of the solution proposed, and the scalability aspects. Conclusions are given in Section 6.

2. The Cloud ICARO Architecture

The proposed Smart Cloud solution addresses some the critical issued put in evidence in the introduction, coping with complex business configurations deployed on the cloud and coming from multiple software providers. Thus, a knowledge base driven solution for smart cloud management is proposed, providing more flexibility and programmability with respects to state of the art and commercial solutions. In most of the cloud management systems a set of configurations are offered to the cloud buyers and produced by a Cloud Configuration Manager, CCM. The business configurations, as well as changes of configurations, are typically deployed on the cloud by using an Orchestrator (for example VCO, MS, etc. as well as other open source solutions), that has also the duty of setting up the monitoring and supervising activities, and in some case also for some smart cloud features, such as in the IBM solution. Most of the commercial Smart Cloud solutions, have limited capabilities in formally defining and detecting complex conditions about resources consumption and status that can be used for activating changes in the cloud, and firing scaling up or down rules (vertical and horizontal, elastic, etc.).

The proposed Smart Cloud architecture is reported in Fig. 1, which includes the Smart Cloud Engine, SCE, with its front-end. The SCE can be invoked and exploited by any CCM or Orchestrator, for: (i) registering new business configurations and their corresponding SLA; (ii) requesting verification and validation of specific business configurations, and receiving back suggestions and hints related to feasibility, consistency and completeness; (iii) activating sporadic and/or periodic monitoring of the SLA associated with a contract; (iv) activating the sporadic and/or periodic control of the healthiness of a business configuration and/or of any specific resource and service on the cloud at any level; (v) enabling the assessment of firing conditions for activating reconfiguration strategies associated with business configurations; for example, for dynamic scaling, VM cloning, VM moving, activating reconfiguration rules. To perform these activities the SCE exploits a: (a) Knowledge Base, KB, in which business configurations, cloud models, SLA and data are registered, (b) a distributed scheduler (Distributed SCE Scheduler) to periodically and sporadically put in execution processes capable to assess conditions, compute possible solutions, provides suggestions to CCM and Orchestrators, (c) one or many monitoring tools (called in the figure Supervisor and Monitor, SM) for collecting data from cloud in massive manner according to different layers: IaaS, PaaS, SaaS, and business configurations.

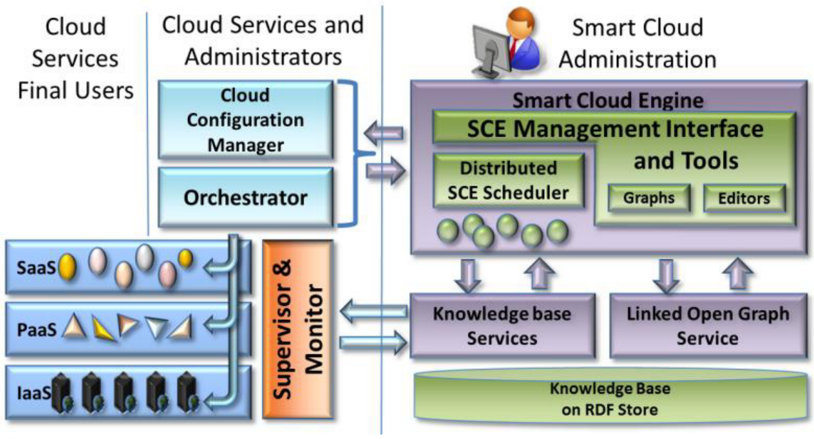

Cloud Services Final Users
Cloud Services and Administrators
Cloud Configuration Manager
Orchestrator
Smart Cloud Administration
Smart Cloud Engine
SCE Management Interface and Tools
Distributed SCE Scheduler
Graphs
Editors
SaaS
PaaS
IaaS
Supervisor & Monitor
Knowledge base Services
Linked Open Graph Service
Knowledge Base on RDF Store

Fig. 1 - ICARO Smart Cloud Architecture

The KB automatically programs the SM services and tools to set up and activate the monitoring processes for controlling services and resources. To this end, SM Service uses drivers to manage multiple Nagios instances (not discussed in this article). This approach has a couple of advantages. Firstly, it simplifies the work on the Orchestrator since all the monitoring configurations and management issues do not have to be programmed into the workflow for deploy of resources, thus reducing the complexity of the orchestration that is an error prone process. In this manner, the SCE automatically adds all monitoring processes and structures that permit at the SCE to have needed information for controlling the business configurations behavior related to SLA, to scaling and reshaping strategies, and of all resources involved. Moreover, the SLAs are typically based on high level metrics as applicative metrics, such as for example: the number of times the workload exceeded a given threshold for 20 minutes, average the number of users registered per day, the number of contemporary streaming processes, etc. Once the SM monitoring processes are activated, the useful measuring data are received and collected into the KB. Other detailed data may be left cumulating data into the monitoring tools and services. In order to fastening the alignment of the Smart Cloud solution with an eventual VMware based infrastructure already in place, the SCE also presents a process to directly access at the vCenter data of a vSphere solution.

The SCE interface allows to easily activate for each business configuration and cloud resource the: (i) strategies (reconfiguration, scaling, cloning, migration, etc.), (ii) verifying healthiness of cloud resource, and of SLA of business processes allocated on cloud, and for (iii) controlling the whole cloud conditions with high level parameters. To this end, the SCE presents a suitable user interface called SCE Management Interface and Tools, which includes (i) editors for programming the Smart Cloud, for the definition and management of SLA monitoring, assessment, and strategies setup, (ii) graphics rendering tools for depicting the trends of metrics and SLA parameters with respect to firing conditions. These activities are performed without requesting to program the processes that are activated by the SCE on the Distributed SCE Scheduler, DISCES. The processes of DISCES can be sporadic and/or periodic and exploit the KB by performing semantic queries in SPARQL. Thousands and thousands of processes are executed per day by the DISCES. That put them in execution on a distributed and parallel architecture comprised of a set nodes (on virtual machines and a distributed scheduler), thus obtaining a scalable and fault tolerant solution for Smart Cloud. If needed, the processes of DISCES can be manually formalized. To this end, for the formalization of semantic queries, a suitable graphical user interface based on Linked Open Graph, LOG (http://log.disit.org), can be used to access the KB and browsing the semantic model, Bellini, Nesi and Venturi, 2014[18].

Please note that, the proposed solution for Smart Cloud solution can be easily integrated with any CCM, and/or Cloud Orchestrators, and by exploiting some monitoring tool since the connection with these subsystems are performed by using REST calls and XML/JSON files. In the validation case, it was directly managed by a higher level CCM addressing different kinds of orchestrators.

3. The Smart Cloud Engine Knowledge Base

The KB collects and organizes the configurations of the whole cloud services ranging from the infrastructure elements to the applications, and also addressing the applicative metric definitions and values. A review on cloud ontologies and KB usage in the context of cloud can be recovered on Bellini, Cenni, Nesi, 2015[19]. The use of a KB facilitates interoperability among public and private clouds, and/or among different cloud segments; allows formal verification and validation of resource cloud configuration, discovering and brokering services and resources, reasoning about cloud security, computing capability for horizontal or vertical scaling, thus implementing strategies for elastic computing. To this end, the KB needs to store not only the cloud structures (infrastructure, applications, configurations, SLA) and also the values of metrics in time to be able to answer questions such as “Which host machines can allocate a new VM with certain features?” or “Which are the overloaded hosts?”, “Which business configuration has violated SLA parameters?”. The storage of the full history of all metric values on the KB can be too expensive and unnecessary. Therefore, in most cases, only metrics related to the SLA and high level metrics are used to have concise indicators instated of low level metrics values (CPU%, memory used, disk available, database size, etc.). The high level metrics can be used to represent the resource load less affected by small sporadic changes (e.g., average or maximum CPU usage % over last hour, % of time over the threshold). On the other hand, values of low level metrics are stored in the monitoring service every 5s, while high level metrics may produce more sparse values, the space needed on the KB is reduced. Regarding applications, in the KB both the application as a type and the application instances need to be stored. Applications may have specific constraints, like the number of services involved (e.g. number of front-end web servers) in order to avoid duplicating the type/instance relation (already modeled in RDF/OWL) and to leverage on the modeling capabilities of OWL2 to express constraints (e.g., max/min cardinality) the application model is represented as an OWL Class.

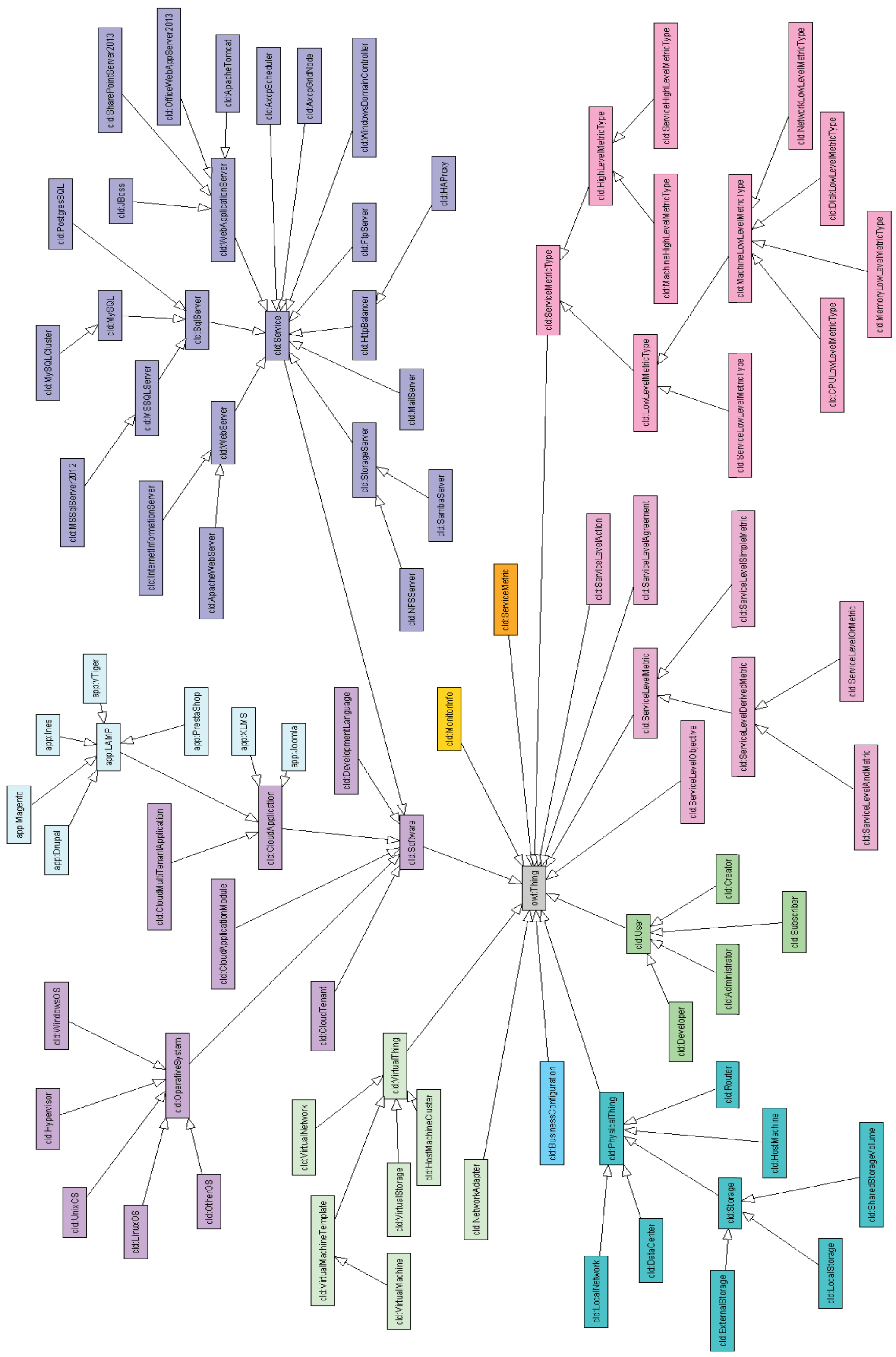


Fig. 2. Cloud Ontology showing only "isa" relationships, the full model can be accessed from the formalization document or browsed from the LOG.

Another need is the possibility to aggregate different applications, servers, virtual machines to build a complete business configuration (e.g., an ERP with a CRM) and also to model applications tenants and to be able to put application tenants in business configurations. The KB also keeps the SLAs associated with business configurations and applications. The SLA is modeled as a set of Boolean expressions that relate metrics values of a component with a reference value (e.g., a VM average CPU use over last 30 min has to be less than 60% and the number of web server processes running on a VM has to be greater than 0). Moreover, in the SLA, it should be possible to use application-specific metrics.

3.1 Smart Cloud Ontology of ICARO

The cloud description ontologies available in literature and mentioned in the introduction (Androcec et al., 2012[14]), are focused on the description of cloud services mainly aiming at cloud services discovery or searching/matching cloud offers with cloud demand. Moreover, the cloud model on KB can be used for controlling configuration and monitoring cloud evolution. For this reason, a cloud ontology has been developed allowing the representation of the different aspects of a cloud service at all levels (IaaS, PaaS and SaaS) as: infrastructure description (e.g., host machines, virtual machines, networks, network adapters), applications and services description, business configurations, metrics and SLA at cloud level and also monitoring aspects. In Fig. 2, the structure of the ontology showing only the sub class relations is reported. The whole ontology is accessible at http://www.disit.org/cloud_ontology/core (as Linked Open Vocabulary) and other details are available at http://www.disit.org/6715. The proposed cloud ontology is grounded on vocabularies as standard OWL, RDFS, and exploits Dublin Core and FOAF for general ontology information. The proposed ontology can be easily combined with other ontologies for Web Services and their related SLA, while it includes the formalization of a cloud service level agreement with computable conditions and references for low and high level metrics directly computable by the SCE. For this reason it was not possible to exploit state of the art SLA models.

The main aspects of ICARO smart cloud ontology are described as follows.

1) **Infrastructure**: The infrastructure is modeled as a DataCenter with a set of HostMachines or HostMachineCluster (containing HostMachines). The HostMachines have specific attributes as the CPU core count, CPU architecture (e.g., x86), CPU speed, the RAM memory capacity, the network adapters, the LocalStorages available, the hypervisor used, etc. The DataCenter can also contain some ExternalStorage that may be used to store virtual machines, and Routers and Firewalls that connect Networks. The HostMachines contain VirtualMachines that are used to run the services providing applications to users. VirtualMachines are characterized by virtual CPU count, RAM memory capacity, mass storage (different disks) and network adapters that connect to the network.

2) **Applications and Services**: Cloud Applications are realized using a variety of services (e.g., web servers, http Balancers, application servers, DBMS, application caches, mail servers, network file servers) that are available on machines over a network. These services may be deployed in many different ways, from all services on a single machine to one machine for each service. There are some constraints to be fulfilled, for example some services are optional (e.g., application caches), some service need a specific feature (e.g., a web server supporting PHP). As already mentioned, to model Applications we have to represent both the application as a ”type” (the class of Joomla applications) and applications as instances of a type. The following is the definition of the CloudApplication class written using Manchester notation:

```
CloudApplication = Software
    and (hasIdentifier exactly 1 string)
    and (hasName exactly 1 string)
    and (developedBy some Developer)
    and (developedBy only Developer)
    and (createdBy exactly 1 Creator)
    and (createdBy only Creator) and (administeredBy only Administrator)
    and (needs only (Service or CloudApplication or CloudApplicationModule))
    and (hasSLA max 1 ServiceLevelAgreement)
    and (hasSLA only ServiceLevelAgreement)
    and (useVM some VirtualMachine)
    and (useVM only VirtualMachine)
```

An application is defined as a piece of software with an identifier, a name, developed by some developer, whose instance was created by one creator and can be administered by administrators. Moreover, it needs Services, other CloudApplications or CloudApplicationsModules, and can have at most one SLA and it uses some virtual machines. The subclass of applications representing the balanced Joomla applications are those that need one MySQL server, one http balancer, one NFS server for storing files and more than one Apache Web server supporting PHP 5, that is expressed as the following:

```
JoomlaBalanced SubClassOf CloudApplication
    and (needs exactly 1 MySQLServer)
    and (needs exactly 1 HttpBalancer)
    and (needs exactly 1 NFSServer)
    and (needs min 1
    (ApacheWebServer and
    (supportsLanguage value php_5)))
```

Moreover, it is possible to define multi-tenant applications where more tenants can be associated with the application instance, and each tenant can be bought separately and can have a specific SLA.

3) **Metrics and SLA**: For the definition and verification of SLAs metrics are compared with reference values. To this end, two kinds of metrics are defined: low level metrics whose values are provided directly from the SM sub-system (e.g., CPU%, memory used, network bandwidth used) that have point measure at a moment in time; and high level metrics that combine the values of low level metrics to provide a measure of a more general characteristics (e.g., the average CPU usage percentage over the last 30 min.). The ICARO smart cloud ontology defines both: low and high level metrics. The high level metric definition can combine the last value or the average, maximum, minimum, sum of values over a time interval (seconds, minutes, hours) of low level metrics using the basic mathematical operators (plus, subtract, multiply, divide). For example, the ratio between the maximum memory used in the last 10 minutes and the average memory used in the last 30 minutes can be defined as a high level metric. Moreover, SLAs can be defined and associated with applications, application tenants or with business configurations. A SLA is defined as a set of ServiceLevelObjectives (SLO) that need to be verified within a certain validity interval. Each SLO is associated with a logical expression that needs to be verified, this expression is the AND/OR combination of checks of values (less than, greater than, equals) of high level metrics with reference values; the SLO is also associated with an action that needs to be performed/started when the objective is not verified.

4) **Business configurations**: The business configurations contains the instances of the applications that need to work together to form a business process (e.g., multitier solution with scalable frontend/backend). Business configurations contain the application instances, the related service instances (application servers, DBMS, file storage, etc.) that are running on virtual machines. Moreover, a business configuration can also contain simple virtual machines that are not used in an application and it can also contain host machines that are fully available for a specific customer. The business configuration can also contain an application tenant with a specific SLA.

5) **Monitoring**: When providing applications, services or host/virtual machines the information that should be used to enable monitoring them may be also specified, for example the monitoring IP address to be used, in case the machine has multiple IPs. Or the parameters to be used for monitoring a specific service metric.

3.2 Validation and Verification via the KB

The KB allows storing and manipulating DataCenters, Application Types, Business Configurations, Metric Types and Metric Values using specific REST services, and SLA. The KB is physically stored in an RDF Store (an OWLIM-SE instance). The KB provides a SPARQL endpoint to perform semantic queries. The data is provided via REST services as RDF-XML files and it is validated to check if it is consistent with the ontology.

As a general consideration, the OWL model for ontologies is designed for distributed knowledge representation and uses the Open World Assumption (OWA), meaning that: something that is not explicitly stated it is unknown if it is true or false. For example, if it is stated that: an application needs at least two web servers, while in the configuration presents only one; then, a simple reasoner will not considered this a contradiction because in principle we do not know if a second web server exists, and neither that does not now exists. This problem can be solved by adding specific information, completing the knowledge. Another aspect is that OWL does not use the Unique Name Assumption, meaning that two different URIs may identify the same thing. For example, if an application must have exactly one DBMS service and in a configuration the application is associated with two DBMS identified with two different URIs, a standard OWL reasoner will not identify an inconsistency, unless it is explicitly stated that the two URI identify different things. For these reasons, for the validation of the configurations submitted to the KB, SPARQL queries have been directly used to check if a configuration is valid, similarly to Sirim and Tao, 2009[20] where some OWL axioms are transformed to SPARQL queries to express integrity constraints. Therefore, for example, a query to list all the VMs in a configuration (each configuration is stored in a different graph) with a non valid operative system is:

```
SELECT ?vm ?os WHERE
{ GRAPH <...>     { ?vm a cld:VirtualMachine; cld:hasOS ?os. }
FILTER NOT EXISTS     { ?os a cld:OperativeSystem. }
}
```

However, in this case a problem arises when the range of the cld:hasOS property is declared an cld:OperativeSystem and this fact is stored in a reasoning-enabled RDF store. In this case, when the configuration with a wrong reference to the operative system is stored the rule based reasoner infers that this wrong OS identifier is an OS and the query will not identify the problem. For this reason, the range declaration should not be present in the ontology. Using SPARQL queries also max/min cardinality can be checked. Using SPARQL queries to validate the configurations allows checking also aspects that using OWL2 language cannot express. For example a SPARQL query may check if the host machine has space for the new virtual machine. Moreover a SPARQL query may be used to find a host where a new virtual machine can be allocated. Since the SPARQL queries are stored in a configuration it provides a high flexibility, the query can be changed when the ontology is changed to address new needs (e.g., support for Linux Containers), this can be done without changing the application code.

4. Smart Cloud Engine, the SCE

A typical major requirement in a Smart Cloud environment consists of a scalable engine for monitoring and making decisions. In this context, the goal is to develop efficient scheduling solutions for Smart Cloud management and scheduling, see for example Sindhu and Mukherjee, (2011[21]); Ma et al. (2013[22]); Tsaia et al. (2013[23]). To this end, a distributed scheduler for putting in execution monitoring and making decision processes has been developed. The DISCES (Distributed SCE Scheduler) is a core component of the ICARO infrastructure that periodically checks the status of the resources in the cloud infrastructure (e.g., virtual machines and application services). DISCES consists of a set of distributed instances of running agents performing concurrent tasks on a set of virtual machines also allocated on cloud. DISCES is connected to the Knowledge Base on which performs SPARQL queries on the basis of the SLA of the specific cloud service and may invoke REST calls toward the CM to get specific monitoring data (when needed). SCE can be used for VM scaling and reconfiguration (VM moving, cloning or migration, memory or disk increasing, enabling load balancing or fault tolerance). Based on cloud reasoning policies, the SCE takes consequent actions upon detecting critical issues in the cloud infrastructure, both at physical or application level. SCE can check both metric and service status at all levels of the cloud stack. It is aware of the current system and cloud configuration and applies general or specific rules for production and scaling, working with an event based logic that allows taking decisions, for dynamic balancing of resources. SCE allows the user to define rules for defining automatic policies, for example for the management of alarms and events, to optimize the resource utilization across various datacenters (e.g., increasing of computational capacity, automatic data migration). SCE includes DISCES multiplatform scheduling engine with cluster functionality that allows adding distributed nodes and defining jobs, without service downtime. Each scheduling job includes a name and a related group, a fire instance id, a repeat count, a start and an end time, a job data map, a job status (i.e., fired, running, completed, success, failed), and some associated triggers with their metadata (i.e., name and group, period and priority of execution).

In a cloud environment, the rapid dynamic changes involving the parameters related to the various services require to define policies to maintain the coherency of data. SCE supports both concurrent a non-concurrent schemes for jobs. SCE allows a direct monitoring of each job activity with a push interface, reporting the current status of the job, and the number of successes or failures in the last day or week, with relative percentages. Furthermore, the variety of the hardware at disposal and the jobs to be scheduled require best practices for adaptive job scheduling. For example, a reconfiguration process written for a particular CPU architecture should be bounded to run on a certain set of scheduler nodes only; nodes with high CPU load could reject the execution of further tasks, until their computation capacity is fully restored at acceptable levels; more in general, there could be the need to assign certain selected tasks only to nodes with a certain level of processing capacity. It is also of fundamental importance to be able to define mechanisms that allow scheduling tasks in a recursive way, based on the results obtained in previous tasks. For example, a reconfiguration strategy consisting of various steps could require taking different actions on the basis of dynamical parameters evaluated at runtime. At this regard, SCE allows adaptive job execution (e.g., based on the physical or the logical status of the host), and conditional job execution, supporting both system and REST calls. The user can build an arbitrary number of job conditions that must be satisfied in order to trigger a new job or a set of jobs, or can even specify multiple email recipients to be notified in case of a particular jobs result. By combining an arbitrary number of conditions, it is possible to define complex flow chart job execution schemes, for the management of different cloud scenarios. A trigger associated to a conditional job execution is created at runtime and it is deleted upon completion. It is possible to define physical or virtual constraints (e.g., CPU type, number of CPU cores, operating system name and version, system load average, committed virtual memory, total and free physical memory, free swap space, CPU load, IP address), that bind a job to a particular schedulers node. Smart cloud best policies require services and tools to collect and analyze huge amount of data coming from different sources at periodic intervals. Virtual machines typically consist of hundreds of services and related metrics to be checked. SLAs often define bounds related to services or groups of services that consist of many applications, configurations, processing capacity or resources utilization. It is worth noting that collecting such a high number of data could lead to unmanageable systems, even if adopting the best practices of DMBS management or clustering, in a short period of time. For this purpose, SCE includes support for NoSQL, with the aim of allowing high performance in data retrieving and processing. SCE includes event reporting and logging services, for a direct monitoring of the smart cloud infrastructure and the activity status of every cluster node, and notifications about the critical status of a system or service (e.g., sending of emails). Notifications can be conditioned or not to the results of execution.

SCE includes a web interface that allows monitoring the status of the cloud platform (i.e., hosts, virtual machines, applications, metrics, alerts and network interfaces), with details about the compliance of metrics with respect of the SLA, and a summary view of the global status of the cluster nodes (e.g., memory, disk, swap). SCE provides graphs of all the relevant metrics in order to perform deep data analysis (see Fig. 3). SCE performs SPARQL queries to the Knowledge Base to check the coherence of the services with respect to SLA and eventually instructs with a REST call the CM to take reconfiguration actions (e.g., increment of storage, computational resources or bandwidth). SCE includes a logging service for registering every event related to the monitored services, and allows adjusting checking periods for each service. SCE allows to define policies to apply in case of misfired events (e.g., reschedule a job with existing or remaining job count), and allows to produce detailed graphs for every metric (grouped per VM or not), with customizable time intervals. SCE reports for each metric the total amount of times it was found to be out of scale, with respect to the total number of performed checks. Logged metrics report the list of SLA violations occurred in the selected time period, with relevant data (e.g., the time at which the violation occurred, the name of the metric, the registered value, the threshold, and the related business configuration, virtual machine and SLA). SCE reports a global view of the cluster status and detailed views of each node. It is possible to monitor parameters such as last job execution time, number of jobs processed since the last restart, CPU load and utilization, time of last check, free physical memory and the total consumed computational capacity of the cluster (e.g., total CPU utilization, total capacity in terms of GHz and percentage of consumed capacity, total and free memory).

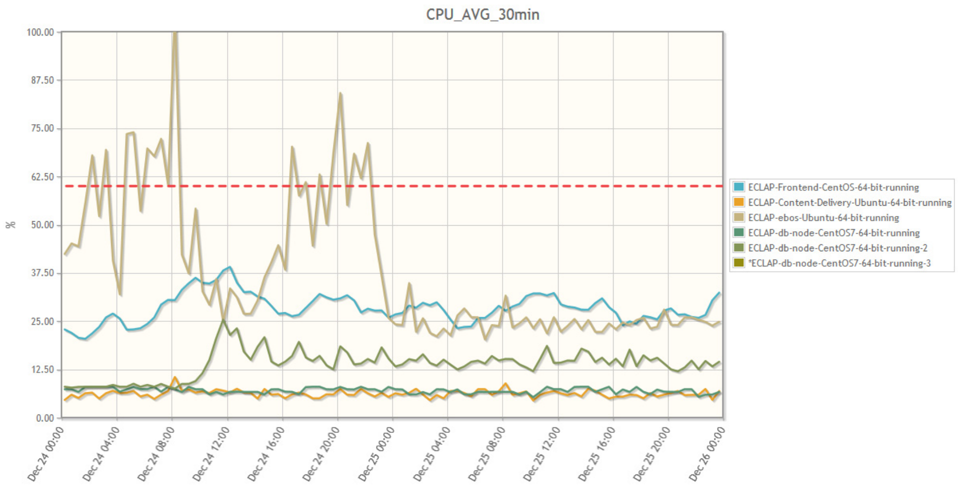
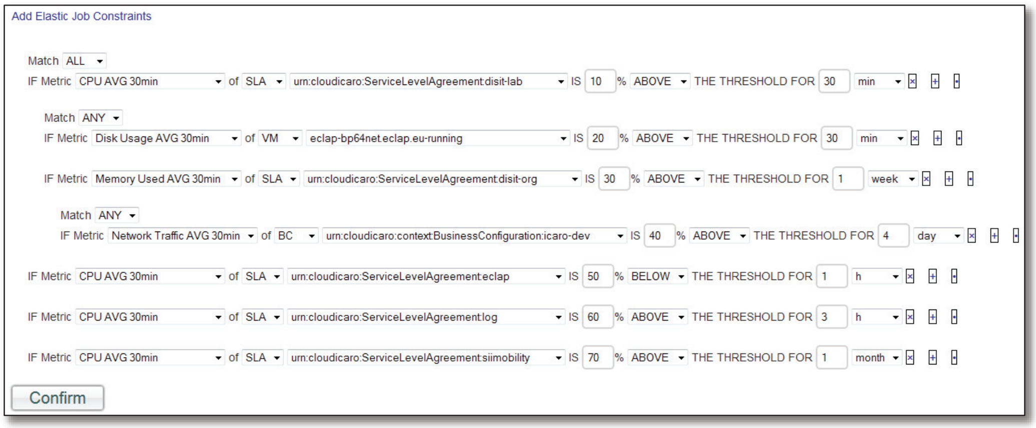


Fig. 3. Metric graphs of a Service Level Agreement, SLA, of a set of VMs

Concerning the best practices of elastic cloud computing, another requirement consists in allowing smart policies for resource scaling and reconfiguration, based on parameters not necessarily included in the SLA. At this regard, SCE integrates a module for the definition of elastic cloud policies, thus allowing defining custom Boolean expressions on metrics, virtual machines and business configurations (see Fig. 4). Thus, it is possible to define complex periodic queries to be performed on the historical collected data, to verify the status of applications and services and to set custom alert limits, related or not to the correspondent SLAs. These periodic tasks can forward REST calls to a specified endpoint, to perform scaling and reconfiguration task on the system or service of interest.

The user is able to define custom elastic policy with the aim of a Strategy Condition Editor. In a typical scenario, it is necessary to monitor several hardware parameters (at physical or virtual level), for example the average RAM and CPU utilization of a host over a defined period of time (e.g., in the last hour). The scheduler engine of the Smart Cloud Engine periodically checks the average value of the requested parameters in the chosen time interval, with respect to what is defined in the Service Level Agreement. Each check contributes as a Boolean value to the whole Boolean expression that is evaluated. The global Boolean expression is evaluated upon completion of all the parameters' checks. If the overall check return an alarm (true) then a REST call is performed, towards a URL defined in the SLA (alternatively, it is possible to define custom calls with parameters). In this way it is possible to perform consistency checks at various levels (e.g., at hardware or application level) to maintain the system within the requested boundaries and constraints. The list of available metrics include CPU, disk, memory, network related parameters, that can be related to different configurations (i.e., SLA, Virtual Machine, Business Configuration). In this sense, the proposed solution proves to be efficient for the smart cloud monitoring and optimization of services at various levels, since it can deal with different aspects of a service platform, in heterogeneous environments and infrastructures.



```
Add Elastic Job Constraints

Match ALL
IF Metric CPU AVG 30min of SLA urn:cloudicaro:ServiceLevelAgreement:disit-lab IS 10 % ABOVE THE THRESHOLD FOR 30 min
  Match ANY
  IF Metric Disk Usage AVG 30min of VM eclap-bp64net.eclap.eu-running IS 20 % ABOVE THE THRESHOLD FOR 30 min
  IF Metric Memory Used AVG 30min of SLA urn:cloudicaro:ServiceLevelAgreement:disit-org IS 30 % ABOVE THE THRESHOLD FOR 1 week
    Match ANY
    IF Metric Network Traffic AVG 30min of BC urn:cloudicaro:context:BusinessConfiguration:icaro-dev IS 40 % ABOVE THE THRESHOLD FOR 4 day
IF Metric CPU AVG 30min of SLA urn:cloudicaro:ServiceLevelAgreement:eclap IS 50 % BELOW THE THRESHOLD FOR 1 h
IF Metric CPU AVG 30min of SLA urn:cloudicaro:ServiceLevelAgreement:log IS 60 % ABOVE THE THRESHOLD FOR 3 h
IF Metric CPU AVG 30min of SLA urn:cloudicaro:ServiceLevelAgreement:siimobility IS 70 % ABOVE THE THRESHOLD FOR 1 month
Confirm
```

Fig. 4. Strategy Condition Editor on the SCE

5. Experiments and Validation

The presented solution has been deployed on the cloud infrastructure of ComputerGross. The most complex SLA consists of 75 conditions with respect to a service application (i.e., http://www.eclap.eu , a social network with scalable frontend and backend, with indexing and searching facilities, automatic content ingestion service, in load balance, a part of that business configuration is shown in Fig.4) using 13 VMs and running 12 services (i.e., 1 HTTP balancer, 3 Web Servers, 1 Apache Tomcat Servlet Container, 1 MySQL database, 1 AXCP Scheduler, 5 AXCP Grid Nodes). The time needed to evaluate a SPARQL query to check the SLA's compliance and to get the current values for the metrics involved is about 30s; for a SLA on a single VM with 4 conditions (with bounds on CPU usage percentage, memory, disk storage and network metrics) it takes about 2.0s. These results were computed after collecting 3 months of data about service metrics, with about 3800 evaluations per metric (see Table I).

Table I – Metrics Recorded on the infrastructure, example of the ECLAP SLA (12/2014-02/2015)

Service Metric Monitored as the average value of 30 minutes on values assessed every 5 minutes	% of overvalues with respect to reference values expected as defined in the SLA	Incidence of overvalues with respect to the total number of measures (time slots) performed for the business configuration
Memory Usage	73.76%	11.86%
Disk Usage	54.03%	8.57%
Network workload	24.29%	3.85%
MySQL DB Size	59.55%	2.12%
CPU Usage	0.07%	0.01%
Apache HTTP response time	0.1%	0.008%
MySQL Connections response time	0.1%	0.004%
Tomcat HTTP response time	0.08%	0.003%

According to Table I, column third, about 26.44% of the assessed time slots were affected by some overvalue, that may be regarded as alarms or critical conditions. The second column reports the percentage of overvalues with respect to reference values defined in SLA. All data were collected with a time period of 30 minutes. The majority of critical overvalues detected in the period were related to memory (11.86%), disk usage (8.57%), network traffic (3.85%), and database size (2.12%). As a general consideration, the ECLAP business configuration experienced a number of changes in the back office. The overload of memory leads to virtual memory usage and thus to reduction of performances and where solved by adding more nodes, while the most critical aspects are on the detection of storage critical conditions that constrain to strong maintenance. Thus, a number of times, additional nodes in the back office for processing media have been added by scaling to reestablish the workload at a reasonable rate.

5.1. Considerations on Scalability

The critical aspects of Smart Cloud engine is the scalability. The scalability regards mainly the costs connected to the monitoring, supervising and managing the cloud structures and resources. Almost all cloud infrastructures are endowed of some monitoring engine for monitoring cloud resources and processes at IaaS and PaaS levels. The state of the art presents a range of them, and almost all are quite scalable since they can manage with a single server several thousands of processes and may be scaled up by replicating them, cluster by cluster. Their scalability is also assured by the fact that the data they collect are typically decimated in the time (passing from a sample every 5s to a sample every hour). In the sense that, they store recent data a high temporal resolution, while older data are progressively reduced mediating them, or subsampling them. This approach can be a problem for a Smart Cloud engine that may need to have high level metrics grounded on historical data of the year or more. To this end, the SCE collects all data and estimates the high level metrics, minimizing the storage needed. In terms of storage, the KB for 6 months of work on a cloud data center cluster with about 120 hosts and a mean of 11 VMs for host, with a SLA every 3.4 VMs and about 35 services and metrics for each VM, lead to have a single non federated RDF store of about 95 millions of triples. The present big data RDF stores as OWLIM and Virtuoso can scale up to billions on federated storages. In these conditions, monitoring VM healthiness, and SLA we have about 288,1 SCE processes per hour that are executed on 3 VMs, for a total of 32Ghz loaded at the 25% in average. The success rate for the jobs scheduled is typically of the 98% in the week. The failures are mainly due to the lack of connection due to the restart of some fail over solution. All of them are recovered at the successive execution and/or retrial, and when this is not possible an alarm is set to the operator. The scaling of the solution can be easily obtained by creating a federated cluster of RDF store, while the scheduler is fully distributed and any new node can be attached at run time without problems of scalability. The recurrent costs of having a SCE solution with respect to a simple monitoring service is about 3 times, that is to have two more severs for each monitoring server. On the other hand, the SCE can save a lot of time for the operators that do not need to monitoring trends and data to making decision about business configuration adaptation according to the SLA.

6. Conclusions and Applicability of the ICARO Solution

This paper presented a smart cloud solution based on an innovative knowledge model, allowing a flexible management of the cloud resources: verification and management of solution for cloud control and restructuring. The solution proposed is scalable and provides smart reasoning and management for cloud platforms and can be applied to different contexts (e.g., hybrid or federated clouds), supporting load balancing and fault tolerance policies, automatic scaling and elastic computing features. It includes a Knowledge Base, KB, for modeling cloud resources, Service Level Agreements and their evolution, and enabling the reasoning on structures by implementing strategies of efficient smart cloud management and intelligence. Via the KM, it provides formal verification tools and intelligence for cloud control and can be easily integrated with any cloud configuration manager, cloud orchestrator, and monitoring tool, since the connections with these tools are performed by using REST calls and XML files. It has been validated in the large ICARO Cloud project with cloud service providers on complex cloud configurations with good performance in terms of low operating workload and scalability. The limits of the solutions are mainly located in the limited complexity of the decision rules that one can formalize directly in the user interface. When complex rules for taking decision are needed the solution can be to realize specific processes in Java that are executed by DISCES. To this end, specific programmers have to be involved. In order to solve this problem a specific user interface based on System Thinking or Goal Model is under design, presently they are directly programmed in Java.

Acknowledgements

The authors want to thank all the partners involved in ICARO, as ComputerGross and LiberoLogico, and the Regione Toscana for funding the project. ICARO has been funded in the POR CReO 2007 - 2013 programme.

References

1. Wu L, Buyya R. "Service Level Agreement (SLA) in utility computing systems". In Cardellini, V., et al. (eds) Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions. IGI Global, USA (2011).
2. Cuomo A, Di Modica G, Distefano S, Puliafito A, Rak M, Tomarchio O, Venticinque S, Villano U. "An SLA-based Broker for Cloud Infrastructures". Journal of Grid Computing, March 2013, Volume 11, Issue 1, pp 1-25, 2012.
3. Pengcheng, X et al. "Intelligent management of virtualized resources for database systems in cloud environment". Data Engineering (ICDE), 2011 IEEE 27th International Conference on. IEEE, 2011.
4. Ward, JS, Barker A. "Observing the clouds: a survey and taxonomy of cloud monitoring". Journal of Cloud Computing 2014.
5. Youseff L, Butrico M, Da Silva D. "Towards a unified ontology of cloud computing". In Grid Computing Environments Workshop, 2008. GCE '08, pages 1-10, Nov 2008.
6. Zhang, M, Ranjan, R, Haller, A, Georgakopoulos, D, Menzel, M, Nepal, S. "An ontology-based system for Cloud infrastructure services' discovery". 8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 , pp.524,530, 14-17 Oct. 2012.
7. Dobson G et al. "QoSOnt: a QoS ontology for service-centric systems,". In Software Engineering and Advanced Applications, 2005. 31st EUROMICRO Conference on, 2005, pp. 80-87.
8. Ghijsen M, van der Ham J, Grosso P, de Laat C. "Towards an Infrastructure Description Language for Modeling Computing Infrastructures". In The 10th IEEE International Symposium on Parallel and Distributed Processing with Applications, 2012.
9. Moscato F, Aversa R, Di Martino B, Fortis T, Munteanu V. "An analysis of mOSAIC ontology for Cloud resources annotation". Federated Conference on Computer Science and Information Systems (FedCSIS) pp.973,980, 18-21 Sept. 2011. http://www.mosaic-cloud.eu/.
10. Pedrinaci, C, Cardoso, J, Leidig, T. [2014] Linked USDL: a Vocabulary for Web-scale Service Trading, 11th Extended Semantic Web Conference [ESWC 2014], Springer. http://linked-usdl.org/.
11. Binz T, Breiter G, Leymann F, Spatzier T. "Portable Cloud Services Using TOSCA", IEEE Internet Computing, May 2012.
12. Binz T, Breitenbücher U, Kopp O, Leymann F. "TOSCA: Portable Automated Deployment and Management of Cloud Applications". In "Advanced Web Services", Springer, 2014.
13. Bernstein D, Vij D. "Using Semantic Web Ontology for Intercloud Directories and Exchanges". In Proc. International Conference on Internet Computing, 2010.
14. Androcec D, Vrcek N, Seva J. "Cloud Computing Ontologies: A Systematic Review". Proc. of MOPAS 2012, The Third International Conference on Models and Ontology-based Design of Protocols, Architectures and Services, Chamonix, France, April 29, 2012.
15. Ludwig H, Keller A, Dan A. Web Service Level Agreement (WSLA) Language Specification, January 28 2003, available at: http://www.research.ibm.com/people/a/akeller/Data/WSLASpecV1-20030128.pdf.
16. Andrieux A, Czajkowski K, Dan A et al. Web Services Agreement Specification (WS-Agreement), March 14 2007, available at: http://www.ogf.org/documents/GFD.107.pdf.
17. Oldham, N, Verma, K, Sheth, A, Hakimpour, F. 2006. Semantic WS-agreement partner selection. In Proceedings of the 15th International Conference on World Wide Web (Edinburgh, Scotland, May 23 - 26, 2006). WWW '06. ACM Press, New York, NY, 697-706.
18. Bellini P, Nesi P, Venturi A. "Linked Open Graph: browsing multiple SPARQL entry points to build your own LOD views". International Journal of Visual Language and Computing, Elsevier, 2014.
19. Bellini P, Cenni D, Nesi P. "Cloud Knowledge Modeling and Management". Chapter on Encyclopedia on Cloud Computing, Wiley Press, 2015.
20. Sirin E, Tao J. "Towards Integrity Constraints in OWL". In Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2009), Chantilly, VA, United States, October 23-24, 2009.
21. Sindhu S, Mukherjee S. "Efficient Task Scheduling Algorithms for Cloud Computing Environment", High Performance Architecture and Grid Computing Communications in Computer and Information Science Volume 169, 2011, pp. 79-83.
22. Ma L, Lu Y, Zhang F, Sun S. "Dynamic Task Scheduling in Cloud Computing Based on Greedy Strategy". Communications in Computer and Information Science Vol 320, 2013, pp 156-162.
23. Tsaia J, Fanga J, Chou J. Optimized task scheduling and resource allocation on cloud computing environment using improved differential evolution algorithm. Elsevier, Computers & Operations Research, Vol 40, Issue 12, 2013, pp. 3045-3055.