CLOUD KNOWLEDGE MODELING AND MANAGEMENT

Pierfrancesco Bellini, Daniele Cenni, Paolo Nesi

University of Florence, Department of Information Engineering, DISIT Lab

pierfrancesco.bellini@unifi.it, daniele.cenni@unifi.it, paolo.nesi@unifi.it

http://www.disit.dinfo.unifi.it (http://www.disit.org)

keywords: ontology, smart-cloud, knowledge base, interoperability

summary

Modeling cloud knowledge can be the basis for enabling a large range of reasoning applications. The modeling of cloud knowledge base includes ontologies for cloud model representation at level of IaaS, SaaS and PaaS with all details regarding cloud resources. Additional aspects related to service level agreements and business metrics are also modeled to allow reasoning on cloud reconfiguration and adaptation to different conditions. The present state of the art shows a number of solutions and standards addressing mainly the problem of cloud interoperability and thus intercloud federation. The most widespread applications areas are related to (i) facilitating interoperability among public and private clouds, (ii) verification and validation of cloud configuration, (iii) discovering and brokering of services and resources, (iv) computing cloud simulation, (v) reasoning and adapting cloud workload conditions, and (vi) reasoning about cloud security. In this chapter, the main aspects of cloud knowledge modeling are presented and discussed by taking into account the state of the art solutions.

1. INTRODUCTION

Almost all relevant infrastructures are using cloud based approaches to manage their resources, and set up high availability solutions addressing different layers such as IaaS, PaaS, and SaaS. Several different vendors are covering different aspects and supporting different services natively into the cloud solutions. Most of them provide specific products addressing only a limited number of features and services. On the other hand, the availability of a wide range of services is often the basis for selecting different cloud solutions. Among the requested services, there is the need of monitoring, changing, moving virtual machines and services in the same cloud for resource optimization and among different clouds to increasing reliability and for migration purposes. To this end, the modeling and formalization of cloud model and information are becoming more relevant to formalize different aspects of a cloud at its different levels: IaaS, PaaS, SaaS, and towards specific resources: hosts, virtual machines, networks, memory, storage, processes, services, applications, etc., and their relationships.

1.1 MODELING KNOWLEDGE

In the past, as to cloud data modeling, some knowledge representation formalisms were introduced, most of them were rooted on simple data structure on description logics to model also more complex relationships. In recent years, with the beginning of the semantic web a new interest has been aroused in the knowledge description formalisms. The W3C introduced several recommendations for the description of information on the web, and in general, information to be interpreted by machines. The base of the produced standards is the RDF (*Resource Description Framework*). With RDF (http://www.w3.org/RDF/) a fact is represented with a triple made of a

subject, a predicate and an object or a data value. Moreover the subject, the predicate and the object are represented with URI (Uniform Resource Identifier). For example, RDF triple:

http://www.example.com/p.bellini, **http://xmlns.com/foaf/0.1/knows ,** *http://www.example.com/p.nesi*

states that the "*thing*" identified by URI *http://www.example.com/p.bellini* "*knows*" the other "*thing*" identified by *http://www.example.com/p.nesi*. The "*knows*" predicate is also defined as an *object property* and it is identified by URI http://xmlns.com/foaf/0.1/knows. This property belongs to the FOAF (Friend Of A Friend) vocabulary defining aspects and characteristics of people and their relations on the web (http://www.foaf-project.org/).

For a more concise writing, a part of the URI can be considered a prefix that identifies the namespace of the thing being described. So for example "*ex*" could be the prefix for http://www.example.com/ and "*foaf*" for http://xmlns.com/foaf/0.1/ and thus the same triple is expressed by:

ex:p.bellini ***foaf:knows*** *ex:p.nesi*

Moreover, it is also possible to state that something belongs to a class of things and also this fact can be represented by a triple. For example, the following RDF triple states that *ex:p.bellini* identifies something that belongs to the class of people:

ex:p.bellini *http://www.w3.org/1999/02/22-rdf-syntax-ns#type* *foaf:Person*

There is also the possibility to associate simple data values (strings, numbers, dates, etc.) with a subject URI. In the following example, family name and given name of *ex:p.bellini*, are provided. Furthermore the *familyName* and *givenName* are called *data properties*:

ex:p.bellini ***foaf:familyName*** *"Bellini"*
ex:p.bellini ***foaf:givenName*** *"Pierfrancesco"*

When two or more consecutive triples share the same subject URI (as in the previous example), we can write:

ex:p.bellini ***foaf:familyName*** *"Bellini";* ***foaf:givenName*** *"Pierfrancesco".*

A vocabulary defines the common characteristics of things belonging to classes and their relations. A vocabulary can be also called "*an ontology*". And, it is defined using the RDFS (RDF Schema) or OWL (*Web Ontologoy Language*). For example, the "*knows*" object property is defined as having as domain and range class *foaf:Person*. When using this information, what can be inferred is that both *ex:p.bellini* and *ex:p.nesi* belong to the class *foaf:Person*. Moreover, the vocabulary states that the class *foaf:Person* is a sub class of a more general class *foaf:Agent*, thus both *ex:p.bellini* and *ex:p.nesi* belong to the class *foaf:Agent*.

OWL is a family of three ontology languages: OWL-Lite, OWL-DL, and OWL-Full. The first two languages can be considered syntactic variants of SHIF(D) and SHOIN(D) description logics (DL), respectively, whereas the third language was designed to provide full compatibility with RDF(S) (Bellandi et al., 2012). The OWL version 2 language proposed by W3C is quite powerful; it allows the definition of disjunctive classes, union and intersection of classes, functional properties, symmetric, transitive properties, minimum and maximum cardinality of the associated elements of a property and other features. OWL 2 is still based on RDF semantics and provides datatype. OWL 2 has three profiles: OWL2 EL, OWL2 QL and OWL2 RL specifically designed and suitable for reasoning with existential quantifications, query formalization and access, reasoning and formalization of rules, respectively (http://www.w3.org/TR/owl2-profiles/).

In order to exploit the information encoded as a set of RDF triples, it can be stored in RDF stores which are optimized for the RDF data management and for the activation of reasoning processes on the collected knowledge. To this end, a specific query language was designed to find information on RDF store reticular information. Thus, the SPARQL (SPARQL Protocol and RDF Query Language, recursive definition (http://www.w3.org/TR/2004/WD-rdf-sparql-query-20041012/)) uses an advanced matching algorithm to match portion of the RDF graph with a specified template. For example, the following query lists names of people known by a person (identified with his email) directly and indirectly through one or more people:

```
SELECT ?n WHERE {
        ?p1 foaf:mbox <mailto:pbellini@unifi.it>.
        ?p1 rdf:knows+ ?p2.
        ?p2 foaf:name ?n.
}
```

Moreover, integrated ontologies can be adopted to enforce capabilities into the model. For example, by exploiting vocabularies (ontology segments) to define properties such as the *FOAF* for people and structures, *Dublin Core* for metadata, *wgs84_pos* for latitude and longitude representation, *OWL-Time* or *TimeOnt* for reasoning about time and temporal aspects, *INDL* for infrastructure and network description, and *QoSOnt* to define Quality of Service aspects.

1.2 EXPLOITING KNOWLEDGE

The main motivations for modeling and using a cloud knowledge base is related to its exploitation for semantic computing and thus for reasoning about cloud (Androcec, 2012). The modeling can be performed by using ontology in OWL and RDF. A Cloud Ontology and Knowledge base consists in an ontology which can be used as model for a big data RDF store including cloud resource configurations and conditions at level of IaaS, PaaS, and SaaS, SLA (Service Level Agreement) of multitier applications and deployments, monitoring data, supporting reselling, brokerage, etc., and real instance data. In the seminal work (Youseff et al., 2008), an approach to create a cloud ontology has been proposed decomposing problems into five layers: applications, software environments, software infrastructure, software kernel, and hardware. The work identified the challenges and verbally discussed the ontology. First attempts to model cloud aspects have been grounded on the several taxonomical and SKOS (Simple Knowledge Organization System) proposed models (Appistry, 2008), (Lairds, 2009), (Hoff, 2009).

A more precise understanding of the effective usage of cloud knowledge modeling exploitation can be taken from the literature analysis where cloud knowledge bases are used for: (i) facilitating interoperability among public and private clouds including automated configurations and deploy (e.g., virtual machine, storage and services cloning or migration); (ii) verification and validation of cloud configuration structures, virtual machine patterns, hosts, etc., against available resources and structures; (iii) services and resources discovering and brokering, including service level agreement, SLA, analysis and matchmaking; (iv) computing cloud simulation for resource and costs planning, prediction and optimization; (v) reasoning about cloud workload conditions estimated by monitoring and needed for decision taking on exploiting resources, thus moving virtual machine, changing resource parameters, negotiating different agreements according to SLA, detecting critical conditions, etc.; (vi) reasoning about cloud security condition and evolution.

A deeper review of the above potential application fields for cloud knowledge is discussed as follows. Open Grid Forum (OGF, http://forge.ogf.org/sf/projects/occi-wg) with its Open Cloud Computing Interface (OGF-OCCI) aims at defining interfaces for unified interface at level of IaaS.

This would allow the creation of an interoperable layer among different vendors, by using an UML model (Unified Modeling Language). In mOSAIC EC FP7-ICT project (Moscato et al., 2011), the cloud knowledge modeling has been addressed with the aim of creating a common model to cope with the heterogeneity of terms used by different clouds vendors, and with the number of standards referring to cloud systems with different terminology. The major issue for cloud interoperability is the lack of standardized APIs, thus the interaction and migration of VMs is a difficult task. The problem of interoperability has been recently addressed by IEEE Project, P2301 - Guide for Cloud Portability and Interoperability Profiles (CPIP), and by IEEE P2302 – draft Standard for intercloud interoperability and federation with the aim of both defining common interoperability protocols among federated clouds and defining configuration, functionalities, and management of inter-cloud interoperability (IEEE 2014).

The problems of service discovering, negotiation and composition of services for cloud infrastructure based on ontological models have been discussed in (Sim, 2011). The proposed solution included a reasoner for similarity analysis and compatibility analysis. In (Dastjerdi et al., 2010), an architecture and solution to provide virtual appliance on demand has been proposed. The idea is mainly derived from the SLA models adopted for grid computing solutions. For the description of SLAs some efforts were made in the past, beginning with WSLA for the definition of SLAs of WebServices (Ludwig 2003).

As to cloud simulation, a significant example is CloudSim (Calheiros, et al., 2011), where several layers of the typical cloud stack can be simulated, including: IaaS, SLA, etc., without using a knowledge base modeling. The solution is fit to simulate simple cloud solutions, but not problems related to the verification and validation of configurations, smart strategies, etc.

The usage of a knowledge base for reasoning about cloud structures and resources, which means automated provisioning and verification of service composition, configuration, optimizations and deployment, can be defined as “Smart Cloud”. This may consist of a set of semantic modeling and computing tools for cloud status reasoning while considering the cloud status and evolution via the Cloud Knowledge Base. The intelligence on smart cloud is enforced by means of a set of algorithms to perform: detection and prediction of critical conditions, verification and validation of configurations (feasibility in terms of consistency and completeness, while taking into account present and possibly available resources), unexpected correlations about facts on cloud evolution, estimation of slack, automated verification of completeness and consistency, verification of compatibility of SLA (service level agreement) against available resources in time, etc.

Currently there are several efforts in building smart cloud solutions grounded on ontology on cloud computing (Androcec, 2012). Most efforts are focused on the description of the services available on the cloud, to allow users to search and compare services (Zhang et al., 2012).

The most interesting projects on this topic are: (i) Linked USDL (Unified Service Description Language http://www.linked-usdl.org/) used by the *FI-WARE* European project (Linked USDL) providing a set of vocabularies for the description of the different service aspects (core service description, SLA, security, price and IPR (intellectual property rights), even though it is focused on service search and discovery, (ii) the mOSAIC project developed a wide ontology covering many aspects from service deployment to service description and it is also focused on cloud service search (Moscato, 2011), (iii) the Icaro Cloud project developed an ontology for the description of both infrastructure and services considering verification and validation of configurations and monitoring information and SLAs (http://www.disit.org/5482), as well.

In addition to the above reported innovative solutions, there are state of the affairs solutions provided by major vendors such as IBM, VMware, HP, Microsoft, etc. and some specific additional tools and plugins that enforce intelligence on the mentioned infrastructure management systems.

2. CLOUD KNOWLEDGE MODELING FOR SMART CLOUD

This section reports the current efforts in modeling Cloud Knowledge; it is focused on the description of the infrastructure, the platform, applications and business processes.

2.1 MODELING IAAS INFORMATION

The IaaS information contains the parts related to the physical structure of a datacenter that is made of Host Machines connected on one or more networks, while hosts may have virtual machines assigned. What follows is a possible example of a datacenter with 100 hosts, one external storage and two firewalls described using the vocabulary being developed for the Icaro project (http://www.disit.org/5482).

ex:datacenter1 **rdf:type** *cld:DataCenter*;
cld:hasName "production data center";
cld:hasPart ex:host1;
…
cld:hasPart ex:host100;
cld:hasPart ex:storage1;
cld:hasPart ex:firewall1;
cld:hasPart ex:firewall2;

Each host machine can have details on the number of CPUs available, the memory size in GB, the disk size in GB, the network adapters and the installed operative system, as it occurs in the next example:

ex:host1 **rdf:type** *cld:HostMachine*;
cld:hasName "host 1";
cld:hasCPUCount 16;
cld:hasCPUSpeed *2.2;*
cld:hasCPUType *"Intel Xeon X5660";*
cld:hasMemorySize 16;
cld:hasDiskSize 300;
cld:hasLocalStorage ex:host1_disk**;**
cld:hasNetworkAdapter ex:host1_net1;
cld:hasNetworkAdapter ex:host1_net2;
cld:hasOS *cld:vmware_esxi***;**
cld:isPartOf ex:datacenter1;

ex:host1_net1 **rdf:type** *cld:NetworkAdapter*;
cld:hasIPAddress "192.168.1.1";
cld:boundToNetwork ex:network1;

ex:host1_disk **rdf:type** *cld:LocalStorage***;**
cld:*hasDiskSize* 300**.**

…

ex:firewall1 **rdf:type** *cld:Firewall***;**

cld:hasName "Firewall 1";
cld:hasNetworkAdapter ex:firewall1_net1;
cld:hasNetworkAdapter ex:firewall1_net2.

Each host machine contains a number of virtual machines; for example, a virtual machine with 2 CPUs, 1GB of RAM, 10GB of disk, one network adapter, with Windows XP professional running on host5, which is described as:

ex:vm1 **rdf:type** *cld:VirtualMachine*
cld:hasName "vm 1, windows xp";
cld:hasCPUCount 2;
cld:hasMemorySize 1;
cld:hasVirtualStorage ex:vm1_disk**;**
cld:hasNetworkAdapter ex:vm1_net1;
cld:hasOS *cld:windowsXP_Prof***;**
cld:isStoredOn ex:host1_disk
cld:isPartOf ex:host1;

ex:vm1_disk **rdf:type** *cld:VirtualStorage***;**
cld:*hasDiskSize* 10**.**

Moreover each element can have associated information needed for monitoring aspects (see section 3.4).

As to the description of infrastructure resources, ontology like INDL-Infrastructure and Network Description Language may be used (Ghijsen et al., 2012). It defines a generic *Node* that is linked with other Nodes through *Interface*s and *Link*s. *VirtualNode*s are used to represent virtual machines running on Nodes. *NodeComponent*s are used to represent the Processing, Memory and Storage components of a Node.

The differences between the two formalizations seem mainly related to the description of networking aspects, namely in INDL it is more detailed though it lacks some details as the IP address.

The mOSAIC Ontology allows to describe some information about the host and the virtual machines (e.g., CPU, memory, storage), but it does not describe how they are connected in the network and how the virtual machines are related with the host machine. Virtual machines are stored in some storage and associated with some hosts or clusters.

2.2 MODELING PAAS INFORMATION

This section reports current efforts in modeling the platform level, while considering the services used to create applications. In the cloud, services are the building blocks used to create more complex and complete applications which can be used by users. In general, an application uses some services, like a database service, a file system service, a mail service, as well as some web servers or web application servers. Generally these services are requested by other specific applications at SaaS level by allocating them on a set of virtual machines. These virtual machines can host and implement more than one service or cooperate with other virtual machines to implement a service (e.g., a DB cluster). Moreover, these virtual machines can provide services for only one specific customer or can be shared among multiple customers. In the latter case, some kind of authentication and service sharing mechanisms are used.

An application can be modeled using specific constraints (e.g., max 4 web servers), the application can be seen as a class containing the specific application instances. A way to represent an application is by defining its relations using the OWL constructs. What is reported below is the general definition of the class of Applications expressed using the OWL2 Manchester syntax (http://www.w3.org/TR/owl2-manchester-syntax):

Application = *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 *Application* or *ApplicationModule*))
and (hasSLA max 1 *ServiceLevelAgreement*)
and (*hasSLA* only *ServiceLevelAgreement*)
and (*useVM* some *VirtualMachine*) and (*useVM* only *VirtualMachine*)

It states that an Application is a Software, which has exactly one identifier and one name, it has been developed by one or more developers (and only by developers!), it has been created (instantiated) by a creator user, it can be administered only by administrator users, it needs only Services, other Applications or ApplicationModules, it has at most one SLA and it uses some virtual machines. Sub classes of the Service are the services running on a virtual machine. A specific application, for example Joomla, is a sub class of Application with some additional constraints:

Joomla SubClassOf *Application*
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*)))

The Joomla class is defined as a subclass of the intersection of the Application class with the classes of things that need exactly one MySQL Server, one Http Balancer, one NFS server and at least one Apache WebServer supporting PHP 5.

A specific instance of the Joomla application is as follows:

ex:Joomla1 **rdf:type** *app:Joomla*;
cld:hasName “Joomla for my business”;
cld:developedBy ex:user;
cld:createdBy ex:u1;
cld:needs ex:mysql1, ex:apache1, ex:apache2, ex:httpbalancer1, ex:nfsserver1;
cld:hasSLA ex:sla1;
…
ex:mysql1 **rdf:type** *cld:MySQLServer*;
runsOnVM ex:vm1;
…
ex:apache1 **rdf:type** *cld:ApacheWebServer*;
cld:runsOnVM ex:vm2;
cld:supportsLanguage cld:php_5;
…

2.3 MODELING SAAS AND XAAS INFORMATION

This section describes the current efforts in modeling the whole service that is provided by the cloud, by considering also the interoperability aspects and the brokerage of services from different clouds, as well as the description of a whole business process.

As shown in section 3.2, an application may be described by its parts (the services being used) and it may also have associated: pricing information, description of the provided functionalities, service level description, and other aspects. When using this kind of information, a third-party can store all such application descriptions and provide a service allowing to search for applications having some functionality (e.g., an ERP, Enterprise resource planning) with some pricing constraints and some other interesting features.

The most interesting project dealing with this kind of information is *Linked USDL.* It allows to describe the pricing information of a service while reusing other popular vocabularies as GoodRelations (Hepp, 2008), Dublin Core (http://dublincore.org) and FOAF, vocabularies. It allows a service to be associated with a *PricePlan* having different *PriceComponent*s which may be based on different *PriceVariable*s. It also allows modeling complex dynamic pricing. The following example is related to a plan to use a service for 5 euro/month:

```
ex:Joomla rdf:type usdl-core:ServiceOffering;
        usdl-price:hasPricePlan ex:joomlaPriceplan.
        …
ex:joomaPriceplan rdf:type usdl-price:PricePlan;
        usdl-price:hasPriceComponent ex:ppc;
        …
ex:ppc rdf:type usdl-price:PriceComponent;
        price:hasPrice [ rdf:type gr:PriceSpecification ;
                gr:hasCurrency "EUR" ;
                gr:hasCurrencyValue "5";
                gr:hasUnitOfMeasurement  "MON"
         ] .
```

Another aspect related with SaaS is multitenancy. This approach consists in the possibility of exploiting only a part of a shared service and not the entire software application. This portion of the service application is defined as tenant, which behaves as if the full application, whereas the service is shared among all the application tenants. In this case, the shared application has a set of tenants that can have specific SLA (e.g., they may use a certain amount of storage, a certain amount of network bandwidth, a certain amount of connections, etc.).

When the business user creates his business process on cloud, he can decide to use different applications that can be connected to share information. A BusinessConfiguration can be described as a set of applications or applications tenants that may have dependencies one another. For example, a business configuration with a Joomla instance and a CRM (Customer Relationships Management) tenant.

2.4 MODELING SLAS AND MONITORING ASPECTS

As to the formalization of SLAs some efforts were made in the past beginning with WSLA for the definition of SLAs of WebServices (Ludwig, 2003). The SLA is described using XML schema and it is very general, thus allowing to define and compose service metrics.

The description defines services (*ServiceDefinition*) on the basis of parameters (*SLAParameter*) which are defined using metrics (*Metric*) and metrics are defined using functions which can use other metrics. A SLA can be associated with some *Obligations* describing the objectives of the service level to be guaranteed.

The WS-Agreement was developed out of the "Grid Resource Allocation Agreement Protocol Working Group" (GRAAP-WG). The *WS-Agreement Specification V1.0*, defined a protocol to define the agreement between two services, and it was published in May 2007 as an *Open Grid Forum Proposed Recommendation.* The specification is composed of a schema for the agreement description, a set of "*port type*" and "*operation*" to manage the agreements life-cycle (including creation, termination and agreement state control). An *Agreement* is made of a *Context* and some *Terms* divided into *ServiceTerms* and *GuaranteeTerms*, the latter including the conditions which need to be guaranteed by the service. Conditions can be specified with a target value for the parameter or using expressions, but the syntax to be used to express these conditions is not specified, any kind of XML or textual representation can be used, thus limiting the description interoperability.

In (Oldham, 2006), beginning with WS-Agreement an ontology for matching service requests and offers has been defined. This ontology uses *QoSOnt* (Dobson, 2005) to define Quality of Service aspects and *TimeOnt* for temporal aspects. This service (SWAPS) is based on semantic technologies as IBM SNOWBASE for ontology management and IBM ABLE for the reasoning and for inference rules. It should be noted that WSLA expressions (which can be easily modeled in OWL) are used to define service conditions.

The European project NextGrid for the definition of an European platform for grid computing has defined a SLA based mainly on WS-Agreement. Moreover, SLAng (Lamanna, 2003) is a different XML schema for SLA definition, which is far less generic than WSLA and WS-Agreement. Moreover, in the context of the *FI-WARE* project, the SLAware model has been proposed. SLAware defines the formal semantic of the SLA by using the Transparent Intentional Logic (a modal temporal logic) while the data model is defined by using UML.

In the context of *FI-WARE* project, the *LinkedUSDL* provides a part related to SLA representation which is much simpler with respect to *SLAware.* On the other hand, it seems that the work on SLAware modeling and maintenance has been stopped. In LinkedUSDL-SLA, the service level profile is associated with some service level that can be a Guaranteed State or a Guaranteed Action. The service level can be associated with a service level expression, representing in natural language, the description of the condition to be met and it is also associated with variables which are taken into account to check whether the condition is fulfilled or not.

In Icaro Cloud, the SLA allows to formalize a set of conditions based on metric values associated with applications, application tenants and/or complete business configurations. The Icaro Cloud SLA model follows a simplified WSLA model, composing complex and/or conditions based on comparing metric values to constant values (e.g., defined in the SLA contract). For example, Figure 1 depicts a SLA to guarantee a response time less than 5 seconds for the Apache http server and a database size less than 1GB. In Figure 1 the SLA is represented as an oriented graph where nodes are subject or object URI and arcs are the properties relating them.

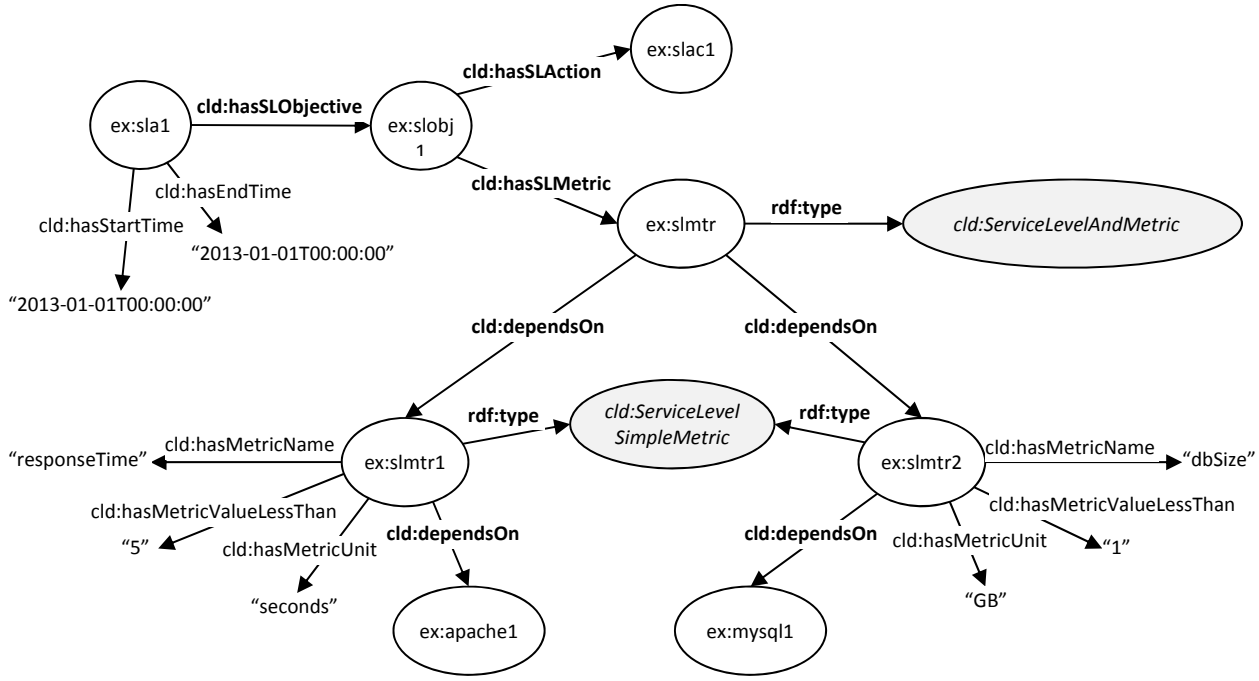

Figure 1. A graph representing a SLA in the framework of the Icaro Cloud ontology

In the MOSAIC ontology, the SLA of a Service allows to associate a set of policies. The latter, in turn can be defined as a set of functional (e.g., monitoring, backup and recovery, replication) and non-functional properties (e.g., CPU speed, network bandwidth, availability). For example, a virtual machine provided as IaaS with x86 CPU architecture, and two CPU cores featuring a High replication can have a SLA represented as follows:

ex:vm_sla **rdf:type** *msc:SLA*;
msc:definedForService ex:vm;
msc:definePolicy ex:vm_policy.
ex:vm **rdf:type** *msc:VirtualMachine*;
msc:hasVirtualizationTechnology msc:Xen;
…
ex:vm_policy **rdf:type** *msc:Policy;*
msc:expressRequirement msc:x86;
msc:expressRequirement [msc:numberOfCPUCores 2];
msc:expressRequirement msc:HighReplication;

The modeling of SLAs can be very different, some approaches are focused on the specification of service metrics bounds and conditions which need to be verified, while others, like Mosaic, are more focused on the specification of high level requirements which are more difficult to be verified. This is due to the fact that, some solutions are oriented to SLA verification/check, whereas others are oriented to allow service search or match. Among solutions focused on SLA checking, there are Linked-USDL and WS-Agreement which do not impose a way to represent the condition which has to be met. On the other hand, other solutions such as WSLA, SWAPS, and Icaro Cloud define specific constructs to represent conditions and fit for their automated computation and reasoning. The latter ones are obviously more suitable for knowledge reasoning.

As far as we know, only Icaro Cloud allows to describe monitoring information associated with host machines, virtual machines, services and applications tenants. The monitoring information may include for example the IP address to be used for monitoring a service or the specific

information regarding the metrics to be monitored; particularly the ones that are specific for the application and that are used in the SLA. For example, an apache web server may have defined a monitor on response time

ex:apache1 **rdf:type** *cld:ApacheWebServer*;
cld:runsOnVM ex:vm2;
cld:hasMonitorInfo ex:minfo1;
…
ex:minfo **rdf:type** *cld:MonitorInfo;*
cld:hasMetricName "responseTime";
cld:has Arguments "http://…"; #specific arguments to be provided to the plugin
cld:hasWarningValue 1;
cld:hasCriticalValue 4;
cld:hasMaxCheckAttempts 3;
cld:hasCheckInterval 5; #check every 5 min

3. SMART CLOUD VS INDUSTRIAL APPLICATIONS

Modeling cloud knowledge can be the basis for enabling a large range of future reasoning applications. Due to the complexity of cloud knowledge models and to the amount of data collected by cloud monitoring systems, the cloud reasoning is becoming a big data problem (Bellini et al., 2013). At the industrial level, one of the closest features to "Smart Cloud" reasoning is the so called: resource optimization tool, elasticity, etc. Elasticity aims to cope with objectives such as: performances, energy consumption, optimization of costs. These approaches could benefit from the presence of cloud knowledge base, and yet in most cases, traditional approaches are used, thus limiting the cloud smartness.

As a general rule, the elasticity is defined as the ability given to customers to quickly request, receive and release as many resources as needed. The *elastic* paradigm in Cloud Computing is strongly related to cloud resource monitoring and prediction, and it should not be confused with scalability, which is the ability of a system to make use of the available increased resources. An elastic application can automatically adapt itself in order to modify the requested or released resources. Therefore, scalability is defined as a static property, and elasticity as a dynamic one. Elasticity policies are divided into automatic (actions are taken on the basis of rules and settings or SLAs) and manual (the user is responsible for the cloud environment monitoring). GoGrid, Rackspace and Microsoft Azure are notable examples of cloud infrastructures where resources are manually managed with no automatic elasticity policies.

Typically, elastic computing includes three different aspects: replication (i.e., horizontal scale), migration and resizing (i.e., vertical scale). Replication includes adding/removing resource instances from the cloud environment (e.g., virtual machines, SaaS modules). Migration includes moving a running virtual machine from one physical server to another. Resizing includes adding/removing processing, memory and storage resources from a running resource instance.

Automatic policies are further divided into reactive (based on rule-condition rules) and predictive. As an example, reactive rules are implemented as it occurs in Amazon. On the contrary, predictive policies make use of heuristics and mathematical techniques to predict the system behavior, and hence to decide the modality and the amount of scaling. **Amazon Web Services** includes a replication feature called Auto-Scaling, in the EC2 service (http://aws.amazon.com/ec2). This feature makes use of the so called Auto Scaling Group (ASG) (i.e., a set of instances at disposal for

an application), and it uses an automatic reactive approach where each ASG includes a set of rules defining the amount of rules that must be added/removed.

4. CONCLUSIONS

The cloud knowledge models reviewed and represented in this paper have been derived from the literature and current aims of standardization. The state of the art of cloud knowledge is presently in evolution. A major effort is needed to fully cover all the potential capabilities of cloud knowledge base applications. The most widespread applications are in the areas of modeling and reasoning about cloud: (i) interoperability among public and private clouds, (ii) configuration at the different level of cloud stack, (iii) service and application discovering and brokering including service level agreement matchmaking, (iv) simulation for workload prediction, (v) dynamic analysis to adapt cloud workload conditions as in the elastic computing paradigm, (vi) security, and security analysis. Most of these application fields need to work on different cloud knowledge models, while a common standard would be needed to make the applications and algorithms interoperable. Modeling cloud knowledge can be the basis for enabling a large range of future reasoning applications. Due to both the complexity of cloud knowledge models and the amount of data collected by cloud monitoring systems, cloud reasoning is becoming a big data problem.

6. REFERENCES

1. (Androcec, 2012) D. Androcec, N. Vrcek, J. Seva, "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.
2. (Appistry, 2008) Appistry. Cloud Taxonomy: Applications, Platform, Infrastructure: http://www.appistry.com/blogs/sam/cloud-taxonomy-applications-platform-infrastructure , 2008.
3. (Bellandi et al., 2012) Bellandi A., Bellini P., Cappuccio A., Nesi P., Pantaleo G., Rauch N., "ASSISTED KNOWLEDGE BASE GENERATION, MANAGEMENT AND COMPETENCE RETRIEVAL", International Journal of Software Engineering and Knowledge Engineering, World Scientific Publishing Company press, vol.32, n.8, pp.1007-1038, Dec. 2012.
4. (Bellini et al., 2013) Bellini P., Di Claudio M., Nesi P., Rauch N., "Tassonomy and Review of Big Data Solutions Navigation", Big Data Computing, Published 26th July 2013 by Chapman and Hall/CRC.
5. (Calheiros, et al., 2011) Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. 2011. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exper.* 41, 1 (January 2011), 23-50. DOI=10.1002/spe.995
6. (Dastjerdi et al., 2010) Amir Vahid Dastjerdi, Sayed Gholam Hassan Tabatabaei, and Rajkumar Buyya. 2010. An Effective Architecture for Automated Appliance Management System Applying Ontology-Based Cloud Discovery. In *Proc. of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing* (CCGRID '10). IEEE Computer Society, Washington, DC, USA, 104-112.
7. (Dobson, 2005) G. Dobson, 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 et al., 2012) Mattijs Ghijsen, Jeroen van der Ham, Paola Grosso and Cees de Laat , “Towards an Infrastructure Description Language for Modeling Computing Infrastructures”, In The 10th IEEE International Symposium on Parallel and Distributed Processing with Applications , 2012.
9. (Guinness et al., 2004) McGuinness, D. L., van Harmelen, F. OWL Web Ontology Language Overview. W3C Recommendation: http://www.w3.org/TR/2004/REC-owl-features-20040210/, 2004.
10. (Hepp, 2008) Hepp, Martin: GoodRelations: An Ontology for Describing Products and Services Offers on the Web, Proceedings of the 16th International Conference on Knowledge Engineering and Knowledge Management (EKAW2008), September 29 - October 3, 2008, Acitrezza, Italy, Springer LNCS, Vol. 5268, pp. 332-347.
11. (Hoff, 2009) C. Hoff. Cloud Taxonomy and Ontology: http://rationalsecurity.typepad.com/blog/2009/01/cloud-computing-taxonomy-ontology.html, 2009.
12. (IEEE 2014) IEEE P2302 -- Standard for Intercloud Interoperability and Federation (SIIF)
13. (Lairds, 2009) P. Lairds. Cloud Computing Taxonomy. In Procs. Interop09, pages 201–206. IEEE Computer Society, May 2009.
14. (Lamanna, 2003) D.D. Lamanna, J. Skene & W. Emmerich, SLAng: A language for defining service level agreements, in Proc. of the 9th IEEE Workshop on Future Trends in Distributed Computing Systems-FTDCS, 2003, pages 100-106.
15. (Ludwig, 2003) H. Ludwig, A. Keller, A. Dan, et al, Web Service Level Agreement (WSLA) Language Specification, January 28 2003, available at: http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf
16. (Moscato, 2011) 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), vol., no., pp.973,980, 18-21 Sept. 2011.
17. (Oldham, 2006) Oldham, N., Verma, K., Sheth, A., and 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. (Sim, 2011) K. M. Sim, — Agent-based Cloud Computing , IEEE Transactions on Services Computing, vol. PP, pp. 1-13, October 2011.
19. (Youseff et al., 2008) Lamia Youseff, Maria Butrico, and Dilma Da Silva. Towards a unified ontology of cloud computing. In Grid Computing Environments Workshop, 2008. GCE ’08, pages 1–10, Nov 2008.
20. (Zhang et al., 2012) 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.

Prof. Pierfrancesco Bellini is professor of Programming Methods of Electronic Calculators, Engineering faculty in the University of Florence. He received his degree in Informatics Engineering in the 1997 and his Ph.D. in the 2001. His main interests and skills are: software engineering, formal methods, data analytics, metadata modeling, LOD, temporal logic, theorem proving, ontologies, distributed systems, object modeling, mobile computing, computer music. He has been involved with EC projects such as ECLAP, AXMEDIS, VARIAZIONI, IMAESTRO, WEDELMUSIC, MOODS, MUPAAC, IMEASY, VISICON, OPTAMS and ICCOC and actually he is involved in industrial projects as: TRACE-IT, RAISSS and ICARO CLOUD. He has been the program co-chair of the WEDELMUSIC, ICECCS and AXMEDIS. He has published several technical papers on international journals and conferences on the above mentioned topics. He has been co-editor of the ISO MPEG SMR standard in MPEG.4.

Dr. Daniele Cenni is currently Research Fellow and PhD in Engineering at the University of Florence. He received the degree in Computer Engineering. His research interests include: indexing systems, collaborative systems, Web Services, Social Networks, User Behavior and Profiling, Information Retrieval. He has participated in European research and development projects like ECLAP and AXMEDIS, and in regional research projects such as: ICARO. He was professor for the course Operating Systems (Master of Science in Engineering, Faculty of Engineering, University of Florence). He has published technical articles in international journals and conferences on the above mentioned subjects.

Prof. Paolo Nesi (http://www.disit.org/nesi/). He is full professor since 2001, obtained the PhD in the University of Padova and conducted a period at the IBM research labs in Almaden (California). Its research skills include distributed systems technologies, formal methods, artificial intelligence, grid and cloud systems, middleware, realtime systems, digital content distribution, smart systems, smart city, intelligent content, e-learning, crawling, data mining. He has been member many international conference committees and editor of international publications and journals. Paolo Nesi published more than 230 articles in international journals and congresses and has been chair and/or programme chair of: IEEE ICSM, IEEE ICECCS, DMS, AXMEDIS, WEDELMUSIC, CSMR, and program committee member of a number of major conferences. Prof. Paolo Nesi is full professor of Distributed Systems in Software Engineering, University of Florence. Prof. Paolo Nesi has been project manager of many big sized European research and innovation projects, like: ECLAP, AXMEDIS, MOODS, I-MAESTRO, WEDELMUSIC, MUSICNETWORK and for the Department in many other projects, like ICCOC, MUPAAC, VISICON, OPTAMS, IMUTUS, IMEASY, ICARO CLOUD, TRACE-IT, RAISSS, and Sii-Mobility. He has been coordinator of the Ad-Hoc Group for the definition of the ISO MPEG-SMR standard and co-author of the ISO MPEG-SMR defined in the MPEG-4 format. Prof. Paolo Nesi is coordinator and responsible of the DISIT Lab of the University of Florence (http://www.disit.dinfo.unifi.it).

//list three or four topics//. Brief info about his/her professional contributions and accomplishments and/or ongoing work. He/She has received //names of awards, fellowships, honors//. For further details, visit his/her webpage //member?s web address, if any// or contact him/her at //email address//.

Pierfrancesco Bellini is professor of Programming Methods of Electronic Calculators, School of Engineering at the University of Florence. His interests include ontology design, formal methods, temporal logics, distributed systems, software engineering. He received a PhD in electronic and informatics engineering from the University of Florence. He has been involved in European Commission projects such as ECLAP, AXMEDIS, VARIAZIONI, IMAESTRO, WEDELMUSIC and currently he is involved in industrial projects as: TRACE-IT, RAISSS and ICARO CLOUD. He has been the program co-chair of the WEDELMUSIC, ICECCS and AXMEDIS conferences. He has been co-editor of the ISO MPEG SMR standard in MPEG 4. For further details, visit his webpage http://www.disit.dinfo.unifi.it/bellini or contact him at pierfrancesco.bellini@unifi.it.

Daniele Cenni is a research fellow and PhD in Engineering at Department of Information Engineering, University of Florence, Italy. His interests include Sharing Systems (P2P), Social Networks, Information Retrieval, Cloud Computing. He has participated in European research and development projects like ECLAP and AXMEDIS, and in regional research projects such as ICARO. He was professor for the course Operating Systems (Master of Science in Engineering, Faculty of Engineering, University of Florence). He has published technical articles in international journals and conferences on the above mentioned subjects. He has received a research grant from University of Florence since 2007. For further details, visit his webpage http://www.disit.dinfo.unifi.it/cenni or contact him at daniele.cenni@unifi.it.

Paolo Nesi is a full professor of Distributed Systems at University of Florence. His interests include distributed systems, smart cloud, knowledge modeling, data mining. He has been member many international conference committees and editor of international publications and journals. Paolo Nesi published more than 230 articles in international journals and congresses and has been chair and/or program chair of a number of international conferences of IEEE, KSI, EC, and program committee member of a number of major conferences. He has been coordinator of a number of large research and development projects of European Commission, and worked for ISO MPEG. For further details, visit his webpage http://www.disit.dinfo.unifi.it/nesi or contact him at paolo.nesi@unifi.it .