Proceedings of the Federated Conference on
Computer Science and Information Systems pp. 973–980

ISBN 978-83-60810-22-4

An Analysis of mOSAIC ontology for Cloud Resources annotation

Francesco Moscato
Second University of Naples
Dep of European and Mediterranean Studies
Caserta, Italy
Email: francesco.moscato@unina2.it

Rocco Aversa
Second University of Naples
Dep of Information Engineering
Aversa, Italy
Email: rocco.aversa@unina2.it

Beniamino Di Martino
Second University of Naples
Dep of Information Engineering
Aversa, Italy
Email: beniamino.dimartino@unina.it

Teodor-Florin Fortiş
Institute e-Austria, Timişoara, Romania, and
West University of Timişoara, Romania
Email: fortis@info.uvt.ro

Victor Munteanu
Institute e-Austria
Timişoara, Romania
Email: vmunteanu@info.uvt.ro

***Abstract*—The easiness of managing and configuring resources and the low cost needed for setup and maintaining Cloud services have made Cloud Computing widespread. Several commercial vendors now offer solutions based on Cloud architectures. More and more providers offer new different services every month, following their customers needs. Anyway, it is very hard to find a single provider which offers all services needed by end users. Furthermore, different vendors propose different architectures for their Cloud systems and usually these are not compatible. Very few efforts have been done in order to propose a unified standard for Cloud Computing. This is a problem, since different Cloud systems and vendors have different ways to describe and invoke their services, to specify requirements and to communicate. Hence a way to provide a common access to Cloud services and to discover and use required services in Cloud federations is appealing. mOSAIC project addresses these problems by defining a common ontology and it aims at developing an open-source platform that enables applications to negotiate Cloud services as requested by users. The main problem in defining the mOSAIC ontology is in the heterogeneity of terms used by Clouds vendors, and in the number of standards which refer to Cloud Systems with different terminology. In this work the mOSAIC Cloud Ontology is described. It has been built by analysing Cloud standards and proposals. The Ontology has been then refined by introducing individuals from real Cloud systems.**

I. Introduction

CLOUD Computing is an emerging model for distributed systems. It refers both to applications delivered as services and to hardware, middleware and other software systems needed to provide them. Nowadays the Cloud is drawing the attention from the Information and Communication Technology (ICT) thanks to the appearance of a set of services with common characteristics which are provided by industrial vendors. Even if Cloud is a new concept, it is based upon several technologies and models which are not new and are built upon decades of research in virtualization, service oriented architecture, grid computing, utility computing or distributed computing ([26], [34], [42]). The variety of technologies and architectures makes the Cloud overall picture confusing [26]. Cloud service providers make resources accessible from Internet to users presenting them *as a service*. The computing resources (like processing units or data storages) are provided through virtualization. *Ad-hoc* systems can be built based on users requests and presented as services (*Infrastructure as a Service*, IaaS). An additional abstraction level is offered for supplying software platforms on virtualized infrastructure (*Platform as a Service*, PaaS). Finally software services can be executed on distributed platforms of the previous level (*Software as a Service*, SaaS). Except from these concepts, several definitions of Cloud Computing exist ([41], [19], [27], [24], [37], [33]), but each definition focuses only on particular aspects of the technology. Cloud computing can play a significant role in a variety of areas including innovations, virtual worlds, e-business, social networks, or search engines but it is actually still in its early stages, with consistent experimentation to come and standardization actions to effort. In this scenario, vendors provide different Cloud services at different levels usually providing their own interfaces to users and Application Programming Interfaces (APIs) to developers. This results in several problems for end-users that perform different operations for requesting Cloud services provided by different vendors, using different interfaces, languages and APIs. Since it is usually difficult to find providers which fully address all users needs, interoperability among services of different vendors is appealing.

Cloud computing solutions are currently used in settings where they have been developed without addressing a common programming model, open standard interfaces or adequate service level agreements or portability of applications. Neglecting these issues current Cloud computing forces people to be stranded into locked, proprietary systems. Developers making an effort in Cloudifying their applications cannot port them elsewhere.

In this scenario the mOSAIC project (EU FP7-ICT programme, project under grant #256910) aims at improving state of the art in Cloud computing by creating, promoting and exploiting an open-source Cloud application programming

interface and a platform targeted for developing multi-Cloud oriented applications. One of the main goal is that of obtaining transparent and simple access to heterogeneous Cloud computing resources and to avoid locked-in proprietary solutions.

In order to attain this objective a common interface for users has to be designed and implemented, which should be able to wrap existing services, and also to enable intelligent service discovery. The keystone to fulfil this goal in mOSAIC is the definition of an ontology able to describe services and their (wrapped) interfaces.

Ontologies offer the means of explicit representation of the meaning of different terms or concepts, together with their relationships. They are directed to represent semantic information, instead of content. Different languages can be considered for the specification of ontologies, including DAML, OIL, RDF and RDFS, OWL or WSML.

The Web Ontology Language (OWL) is a standard from [32], [18], based on XML, RDF and RDFS. With OWL complex relationships and constraints can be represented in ontologies. With important revisions to the language, OWL 2 became the W3C recommendation in 2009, introducing features to improve scalability in applications. [25]

Different efforts to formalize Semantic Web developments exist. Web Service Modeling Ontology (WSMO) [14] "*provides the conceptual underpinning and a formal language for semantically describing all relevant aspects of Web services in order to facilitate the automatization of discovering, combining and invoking electronic services over the Web*" [40]. WSML was offered as a companion language to WSMO, for representing modelled ontologies by a common terminology for Web Services interactions [22], [40]. The Semantic Web Services Framework (SWSF) offers a similar approach, with its two major components, the Semantic Web Services Language (SWSL) and the Semantic Web Services Ontology (SWSO) [17].

Semantically-enabled services offer the means for intelligent selection of services, with automation of different tasks, including service discovery, mediation, invocation, or composition. Current research efforts are enhancing typical web services technologies in order to provide a semantically-enhanced behaviour in developments like OWL-S [30], WSDL-S and METEOR-S [38], [36], WSML [22], WSMO [40], or SWSF [17].

II. mOSAIC Project

The Open Cloud Manifesto [10] identifies five main challenges for Cloud: data and application interoperability; data and application portability; governance and management; metering and monitoring; security.

Actually, the main problem in Cloud computing is the lack of unified standards. Market needs drive commercial vendors to offer Cloud services with their own interfaces since no standards were available at the moment. Vendors solutions have arisen as commonly used interface for Cloud services but interoperability remains an hard challenge, like portability of developed services on different platforms. In addition vendors and open Cloud initiatives spent few efforts in offering services with negotiated quality level.

The mOSAIC project tries to fully address the first two challenges and partially addresses the next two ones by providing a platform which:

- enables interoperability among different Cloud services,
- eases the portability of developed services on different platforms,
- enables intelligent discovery of services,
- enables services composition,
- allows for management of Service Levels Agreement (SLA).

The architecture of mOSAIC platform is depicted in Fig.1: it provides facilities both for end-users (at the left of Fig.1) and for services developers and managers (depicted on the right side of Fig.1)

From the end-users' point of view, the main component is the *Cloud Agency*. This consists in a core set of software agents which implement the basic services of this component. They include:

- negotiation of SLAs;
- deployment of Cloud services;
- discovery and brokering of Cloud services.

In particular, *Client Agent* is responsible for collecting users' application requirements, for creating and updating the SLAs in order to grant always to best QoS. The *Negotiator* manages SLAs and mediates between the user and the broker; it selects protocols for agreements, negotiates SLA creation, and it handles fulfilment and violation. The *Mediator* selects vendor agents able to deploy services with the specified user requirements; it also interfaces with services deployed on different vendors' providers. The *Provider Agent* interacts with virtual or physical resources at provider side. In mOSAIC the Cloud Agency was built upon the MAGDA [16] toolset, which provides all the facilities to design, develop and deploy agent-based services. The *semantic engine* uses information in the Cloud Ontology to implement a semantic-based Cloud services discovery exploiting semantic, syntactic and structural schema matching for searches.

In the Cloud developers and managers perspective, the main components of mOSAIC Architecture are the *API execution engine* and the *Resource Manager*. The first one offers a unique API to use Cloud Services from different vendors when using and developing other services. The API execution engine is able to wrap storage, communication and monitoring features of Cloud platforms. In particular, Virtual Clusters (VC) [23] are used as resource management facility. They are configured by software agents in order to let users to configure required services. A *Resource contract* will grant user's requirements and the Resource Manager will assign physical resources to VC on the basis of the contract.

In this architecture, the bonding element which allows for interoperability and resources description is the *Cloud Ontology*. It is the base for Cloud services and resources description and it contains all information needed to characterize API also from a semantic point of view.

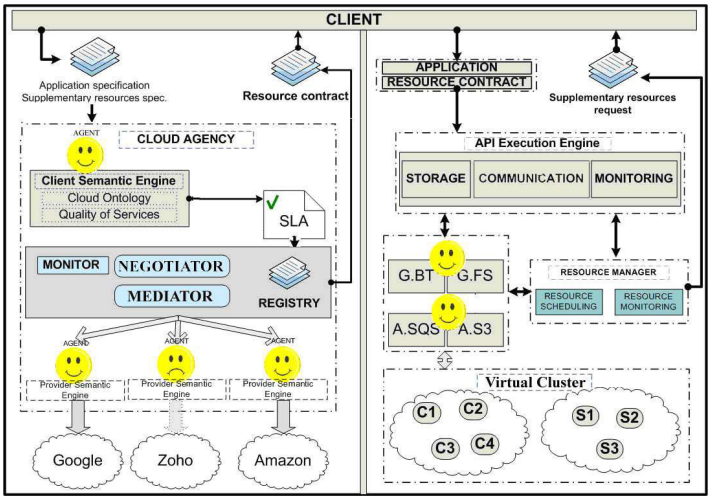


Fig. 1: mOSAIC Architecture

The Cloud Ontology is based on several Cloud taxonomies proposed in literature [15], [4], [28], [20], [29]. It is developed in OWL [32] and OWL-S languages [30]. The benefit of using an ontology language is that it acts as a general method for the conceptual description or modelling of information that is implemented by actual resources [3]. mOSAIC aims at developing ontologies that would offer the main building block to describe services at the three delivery models of Cloud Computing (i.e. IaaS, PaaS, SaaS).

III. Cloud Standards and mOSAIC

Nowadays several Cloud computing systems are available, both from commercial and open source communities. Some example are Amazon EC2 [1], Google's App Engine [6], Microsoft Azure [21], GoGrid [5], 3Tera [7], Open Nebula [12], Eucalyptus [35] and Nimbus [2]. Cloud systems and services offered by various vendors differ and overlap in complicated ways. Each solution provides different services. For example Amazon EC2, GoGrid, 3Tera, Open Nebula and Eucalyptus are basically IaaS Clouds offering entire instances of virtual machines to customers; Google's Apps and Microsoft Azure offer SaaS applications also providing API for development and monitoring, offering a PaaS Cloud. Nimbus was developed as an IaaS for scientific applications. Main vendors platforms and services have become standards *de facto* for Cloud computing, but several different solutions exist at different Cloud layers and interoperability is still a distant goal. In this scenario some attempts have been done to make order in the chaos of Cloud systems trying to propose standards for them.

Anyway the need for a good, complete definition of Cloud components is really felt by scientific community. In particular, in [43]the need for an Ontology defining Cloud- related concepts and relationships is outlined. In the paper an ontology for Cloud is proposed *in natural language*. Cloud layers have been defined and organized in an architectural view. The ontology starts with firmware and hardware as its foundation, eventually delivering to Cloud applications. The paper also defines elements which belongs to different layers, like resources, virtual machines etc. Anyway no formal representation of the Ontology is reported in the paper.

Another similar taxonomy for Cloud Systems has been presented in [39], where only Cloud layers and some requirements like fault tolerance and security have been discussed.

A more detailed Taxonomy has been described in [20]. It is a simple taxonomy, with only main concepts related to Cloud Computing defined in a graphical schema. This work in progress is anyway more complete then the previous ones.

One of the few attempts to provide a formal ontology for Cloud System comes from Unified Cloud Interface (UCI) Initiative that have released a very simple OWL ontology [13] for Cloud Systems but it consist only of few concepts.

Cloud Computing Interoperability Forum (CCIF) [9] aims at defining an open, vendor neutral and standardized Cloud interface for the unification of various Cloud APIs. This should be done creating an API wrapping other existent APIs. CCIF proposes to define an OWL/RDF ontology to describe a semantic Cloud data model in order to address Cloud resources uniquely. The ontology is still under development and no draft version are available at the moment.

Similarly, Open Grid Forum (OGF) [11] is another open initiative which aims at the creation of a practical solution to interface existing Cloud IaaS. OGF is defining interfaces (the Open Cloud Computing Interface: OCCI [11]) to provide unified access to existing IaaS resources. The main goal of OCCI is the creation of hybrid Clouds operating environments independent from vendors and middlewares. The main formalism used to define Cloud models is UML and the work is still in a preliminary stage. OCCI's documents defines specifications for cloud core elements and interfaces to resources, including a model using a RESTful API to access, use and manage them. In OCCI core model, *everything is a resource*. The main elements of the base OCCI model are: Entity, Resource, Link and Action. The Entity is an abstract type for

Resource and Link type; Resources identify object in cloud environment, while Link are used to specify relationships among Resources. Actions define operations applicable to Entities. The OCCI model is developed in UML, but the main elements of the model are used to describe a graph structure that is similar to an OWL ontology definition. The model is not yet complete and several properties and relationships between Entities cannot be described.

The National Institute of Standards and Technology (NIST) is also working at Cloud standards definitions. NIST main aim in defining cloud standards is to provide specifications about interoperability, portability and security requirements, standards and guidance. As part of the NIST plan, a reference Architecture and Taxonomy standards have been proposed. In addition, a roadmap to address security in Cloud systems has been formalized. The NIST definition of cloud computing architecture, includes basically five essential characteristics (on-demand self-service, broad network access, resource pooling, rapid elasticity, measured Service), three service models (IaaS, PaaS, SaaS) and four deployment models (public, private, community and hybrid cloud). Furthermore NIST proposes a taxonomy, organized in layers, where main cloud concepts are organized. In the first level roles for cloud are identified: service provider, consumer, broker, auditors and carriers are listed. They are related to usage scenarios analysed by NIST that focuses on interoperability of federating cloud providers. At second level, activities that actors in the first level enact are defined, while components are addressed in the last two levels. The mOSAIC Ontology, which will be described in detail in Section IV inherits most of the elements defined in other proposals, in order to maintain an high degree of compatibility with them. Anyway, in the NIST taxonomy, too much Roles (Actors in mOSAIC) have been reported and some of them are not easily distinguishable. In addition The taxonomy proposed by NIST is not really a hierarchy since elements in lower layers are not always specialization of upper layer elements. For this reason, mOSAIC inherits only the main actors definition from NIST proposal, and does not maintain the taxonomy proposed by NIST when no *subclass* relationship exists between classes.

Distributed Management Task Force (DTMF) proposes a standards incubators in order define a set of architectural semantics that unify the interoperable management of enterprise and cloud computing. DTMF mainly addresses use cases and cloud reference architecture, analysing the interfaces between cloud service providers and consumers. In use case definition, a key role has the definition of Service Level Agreement (SLA). The interactions between services consumers and providers are detailed. Actors in DTMS proposals are similar to Roles for NIST definition. DTMS also addresses interfaces and data artefacts as a mean for interfacing actors.

Recently IBM [31] has provided a draft document describing a reference architecture for Cloud systems. It recalls the NIST and OCCI standards, integrating some parts with new elements. In particular, the IBM architecture adds Business processes as element of Cloud Architecture. Business Process

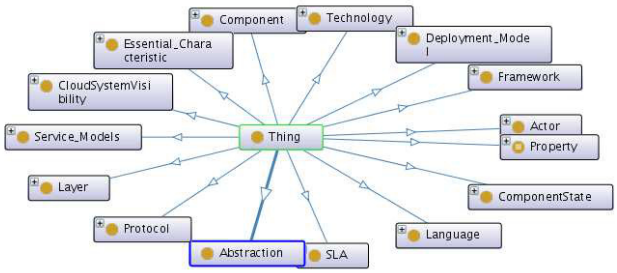


Fig. 2: Top Level Concepts in mOSAIC Ontology

has been introduced as Cloud architecture layer and as new class of actors that can use them in order to create composed services. Proper Business Support Services (BSS) have been defined in order to support business process definition and execution. Furthermore, IBM architecture inherits NIST actors taxonomy, maintaining three main types of actors: services Consumer, Provider and Creator. It also addresses the problem of defining Quality of Services and SLA.

Microsoft Azure [21] and Google APP Engine [6] provide an environment for developing and deploying services and applications following the Cloud philosophy. Azure offers a Platform as a Service (PaaS) and an Infrastructure as a Service (IaaS) hybrid platform on which developers implements and deploy their services. A similar approach is adopted by Google APP Engine. They provide almost no attempt at standardization, except for a small OWL ontology (no more supported) developed by Google. Amazon with its Amazon Web Services (AWS) offers IaaS and SaaS services. It provides, a set of API for creating virtual machines and resources, and to access its services. mOSAIC ontology is able to describe these APIs as will be shown in Section IV-A

IV. mOSAIC Ontology

The top level of the mOSAIC Ontology is shown in Fig.2 which reports the main concepts of the mOSAIC ontology. Concepts have been identified analysing standards and proposals from literature. In the following its main concepts will be listed and described.

The **Language** class contains instances of languages used for APIs implementation (for example, Java and Python). **Abstraction** class contains the abstraction level at which services are provided as described in[43]. Here, Cloud services belong to the same layer if they have equivalent level of abstraction. **Deployment Model** class includes concepts required by Cloud NIST [43] standard for what deployment model of Cloud services concerns. **Essential Characteristics** class includes individuals which are defined by NIST. **Framework** class contains individuals that identify programming framework supporting API programming Languages. **Actor** contains subclasses where actors interacting with Cloud systems are divided. **Property** subclasses contain all elements needed for describing characteristics of Cloud resources. These are also

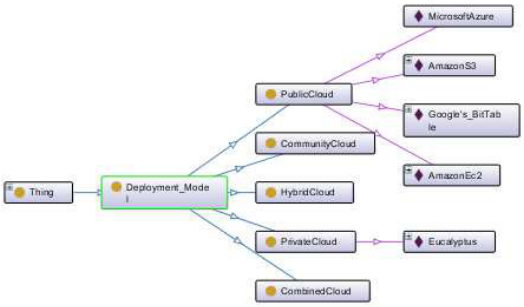


Fig. 3: Deployment Model

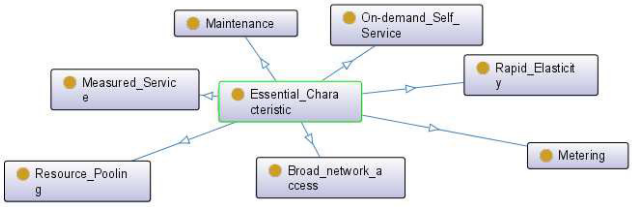


Fig. 4: Essential Characteristics

used to specify SLA requirements. **ComponentState** includes all concepts for defining the states which Cloud components and resources may assume. **SLA** class defines concepts for SLA definitions. **Protocol** class contains individuals for protocols used in communication among Cloud components. **Layers** class distinguishes firmware, hardware and software infrastructures for Cloud platforms. **Service Models** class includes all kinds of services provided by Cloud Systems. **Predicate** contains classes used for description of the behaviours of statefull Cloud components. **CloudSystemVisibility** class allows for specification of Cloud systems visibility, like private and public clouds. **Component** is the main class of mOSAIC ontology. All cloud elements (resources, services, infrastructures etc.) are its subclasses. **Technology** class contains all concepts related to technology involved in Cloud services provisioning, like virtualization.

Fig.3 shows the *Deployment_Model* subclasses.

They include several types of deployment models for Cloud Systems: PublicCloud contains all individuals providing public or world wide access to their resources, like MicrosoftAzure, Amazon and Google. PrivateCloud instead is related to Deployment Models of framework that can provide access to private Cloud resources, like Eucaliptus.

Essential_Characteristics are defined in the NIST standards as the features that each Cloud system must provide. They are shown in Fig. 4.

This Class contains subclasses related to the characteristics described into NIST document about Cloud features (Maintenance, On demand self service, Rapid Elasticity, Metering, Broad Network Access, Measured Services and Resource pooling). For further description of this properties, refer to NIST [8] and IBM [31] standards documents.

The Actor class identifies cloud actors, that can be divided as in Fig.5.

Provider, Consumer and Creator subclasses follow the IBM cloud computing reference Architecture [31]. Administrator manages cloud infrastructure; Orchestrator composes Cloud Services in order to provide value added services; Developer implements new Cloud Services. Notice that the difference between Developer and Creator, is in the way they interact with cloud Providers. Developers use offline resources (tools and frameworks) in order to implement new Cloud Service. A Creator instead builds cloud services by using functionalities exposed by a Cloud Service Provider. Consumer Actors can be further divided as shown in Fig. 5b.

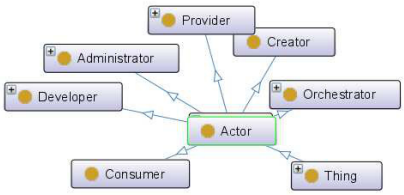
Some Property's subclasses are shown in Fig.6. They are divided into NonFunctionalProperties and FunctionalProperties that respectively define the sets of non functional and functional properties of a Cloud Component. Properties can be used to characterize Cloud Components (services, infrastructure etc.) and to request given characteristics for components when dealing with SLA.

The main non-functional properties for cloud components are: Scalability; Autonomy; Availability; QoS; Performance; Consistency; Security; Reliability.

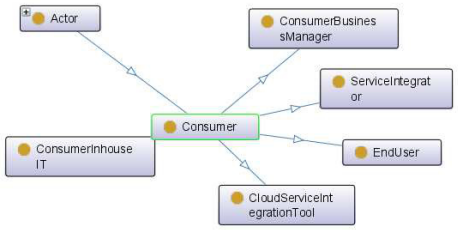
Computing Non Functional properties can be divided into CPU and Memory related properties. A deeper division identifies: CPUSpeedProperty; CPUNumberOfCores; CPUArchitecture; CPUTypeProperty and CPUFlopsProperty. These properties are used to specify the clock frequency, the number of cores, the architecture, the model and the FLOPS of CPUs respectively. The properties follow the OCCI [11] standard and API. A Data Property is defined for each of them in order to specify the value of the property for the related individuals. Properties for memory are divided into: MemoryAllocationProperty and MemorySize. The first property is used to specify memory allocation policies while the second one is used to declare (or require) the amount of memory in a Cloud infrastructure.

Subclasses of this Network Non Functional element are: NetworkLatencyProperty, NetworkDelayProperty, NetworkBandwidthProperty. The first class is used to define the mean latency of a network, the second one the mean, the maximum and the minimum delay for packets and the last one is used to define the mean and the maximum bandwidth of a network. The values for individuals are defined by specifying proper data properties defined on these classes. Data non functional properties are related to disk size (DiskSpaceProperty), transfer rate (DiskTransferRateProperty) and bandwidth (DiskBandwidthProperty).

The main functional properties are: Replication (for the definition of the type of replication policies of resources); Encryption (it specifies the encryption policies of resources); BackupAndRecovery (it is used to describe the back up and recovery strategies used for a Cloud Component); Accounting (its individuals define the accounting policies for

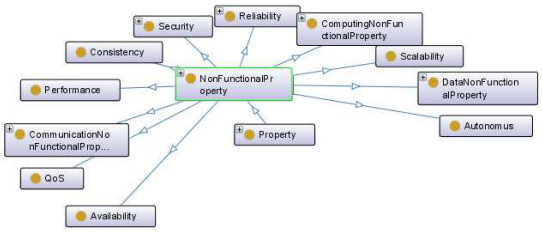


(a) Actor

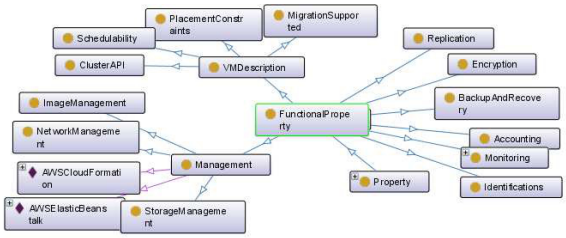


(b) Consumer

Fig. 5: Actors



(a) Non Functional Properties



(b) Functional Properties

Fig. 6: Properties

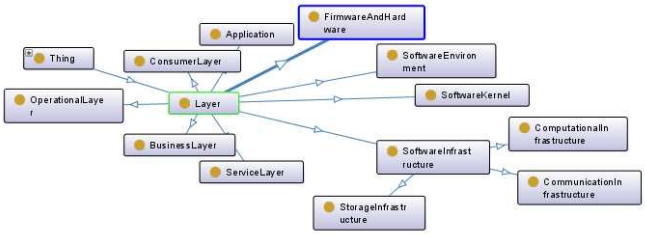


Fig. 7: Layer

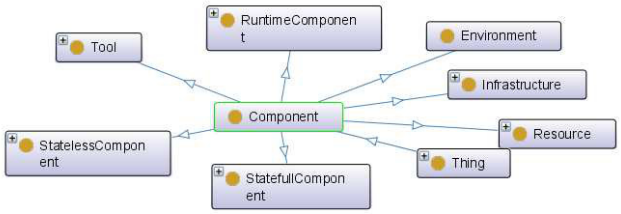


Fig. 8: Component

resources); Monitoring (this class allows for the specification of monitoring policies for resources); Identification (it contains individuals that can specify the algorithms and policies for users identification); VMDescription (used to describe virtual machines technologies and configuration eventually used in cloud infrastructure); Management (it defines the management policies for cloud resources).

Management contains the following subclasses: ImageManagement, NetworkManagement and StorageManagement. The first one is used to define the management policies of a VM image, the second one to define network management policies in a cloud infrastructure, while the third one is used to define storage management policies for cloud resources.

Layers are organized as in Fig.7.

The classes OperationalLayer, ServiceLayer, BusinessLayer, and ConsumerLayer are defined in the IBM Cloud Computing Reference Architecture. They respectively represent: the operational infrastructure layer of cloud systems; the services layer in clouds; the business processes that participate in Cloud solutions; the layer with cloud services and resources consumers.

Furthermore, other layers are derived from OCCI and NIST definitions. They are: Application (the layer where applications lie); Firmware and Hardware (the layers with hardware and firmware of cloud infrastructures); Software kernel (the software kernel for operating systems and middlewares used in cloud infrastructures); Software infrastructure (the layer of computational, storage and communication infrastructures).

Service_Models subclasses includes all models for services in Cloud. Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Service as a Service (SaaS) are the classical models defined in the NIST standard, while the last, BPaaS (Business Process as a Service) is defined in the IBM Cloud Computing Reference Architecture.

Component Subclasses are reported in Fig.8.

They are divided into: Tool (this contains all tools used for

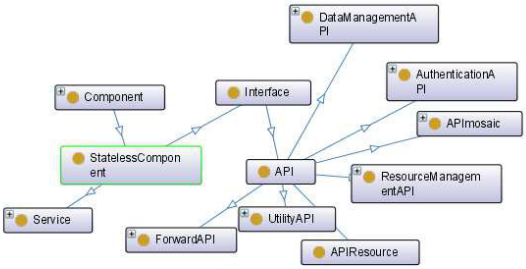


Fig. 9: Stateless Component

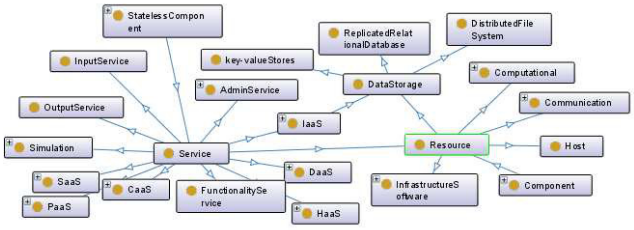


Fig. 10: Resource and Services

TABLE I: ObjectProperties for Individuals

Domain	ObjectProperty	Range
mOSAICDataStore	developedWithLanguage	JAVA
mOSAICDataStore	fulfils	HighReplication
mOSAICDataStore	fulfils	HighConsistency
mOSAICDataStore	hasServiceProvidedby	Google
mOSAICDataStore	isOfferedByProvider	Google
mOSAICDataStore	isOwnedBy	Google
Google	guarantee	HighReplication
Google	guarantee	HighConsistency
Google	guarantee	LowConsistency
Google	offersAPI	GoogleAppAPI
Google	own	mOSAICDataStore
GoogleAppAPI	developedwithLanguage	JAVA
GoogleAppAPI	developedwithLanguiage	Python
GoogleAppAPI	developedwithLanguiage	Go

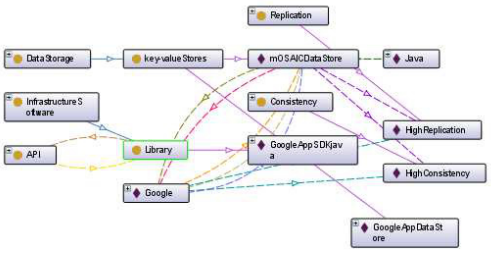


Fig. 11: mOSAICDataStore main Individuals and Relationships

Cloud services development or cloud resources management); RunTimeComponent (this class contains the elements for defining mOSAIC run-time components); Environment (used to define individuals concerning the cloud environment used by cloud services); Infrastructure (describes the component in the cloud infrastructure); StatefullComponent and StatelessComponent; Resource (it collects all resource classes in the cloud ontology).

StatelessComponent class is expanded in Fig.9. Basically stateless component in the ontology are Services and Interfaces. Services will be described later and a general taxonomy for APIs is reported in the figure.

Stateful component are omitted for brevity.

The Resource class is the most complex in the mOSAIC, since, following the OCCI documentation, in Cloud Systems everything is a cloud Resource. Hence this is a super class for other main cloud components as shown in Fig.10.

Service is a Resource. Platform as a Service (PaaS), Computing as a Service (CaaS), Data as a Service (DaaS), infrastructure as a Service (IaaS), Hardware as a Service (HaaS) and other service models (Simulation, services which offers some functionalities, admin, data input and output services) are subclasses of Services. For example, in Figure, DataStorage is provided as an IaaS. Key-valueStores, ReplicatedRelationalDatabeses and DistributedFileSystems are examples of DataStorage services. Cloud Component are also considered as Cloud Resources, like Hosts, Computational and Communication resources or InfrastructureSoftware.

A. DataStorage Example

In this section a brief example is shown where the mOSAIC ontology is used to describe a simple data-storage service implemented with Google App Engine.

The name of the realized service is *mOSAICDataStore* and it implements a key-based data store by using the JAVA Google APP SDK. As described in the Google App documentation, Google Data Stores offer Replication functionalities. In particular, they allow for the use of Master-Slave and High Replication techniques. In addition Consistency on replicas is assured in most cases, although full consistency is not assured. Google offers a framework for development of cloud applications, which is based on an Eclipse Plug-in. The mOSAICDataStore was developed by using this plug-in.

First of all, *Google* has been defined as individual of InfrastructureProvider, DeveloperProvider, ServiceProvider and ResourceProvider; *GooglePluginForEclipse* has been added as a Framework's individual. *GoogleApps* has been included as PaaS and SaaS service Model; *GoogleAppAPI*, an individual representing the Google APP API has been inserted into API, and *GoogleAPPSDK* as Library.

Table I reports some of the objectProperties defined on individuals.

In Fig.11 some of the individuals and the relationships reported in Tab. I are depicted.

V. Conclusions

In this work we propose a detailed ontology for Cloud systems that can be used to improve interoperability among existing Cloud Solutions, platforms and services, both from end-user and developer side. The ontology has been developed in OWL and can be used for semantic retrieval and composition of Cloud services in the mOSAIC project. Several attempts have been done in the past to introduce a Cloud ontology. The ontology presented in this paper also maintains compatibility with previous works because it is built upon

existing standards and proposals analysis and it results in a more comprehensive description of all Cloud-related aspects. The Ontology has been populated with individuals from real Cloud Systems services and APIs and new individuals and elements are going to be included in the ontology with an incremental design approach.

ACKNOWLEDGMENT

This work has been supported by the mOSAIC project (EU FP7-ICT programme, project under grant #256910). We want to thank Daniel Bove for his role in developing the ontology.

REFERENCES

[1] Amazon Elastic Compute Cloud (Amazon EC2): http://aws.amazon.com/ec2/.
[2] Cloud Computing Interoperability Forum: http://www.cloudforum.org.
[3] Cloud Computing Interoperability Forum, Unified Cloud Computing: http://code.google.com/p/unifiedcloud/.
[4] Cloud Computing Interoperability Forum,Cloud taxonomy : http://groups.google.com/group/cloudforum/web/ccif-cloud-taxonomy.
[5] GoGrid: Scalable Load-Balanced Windows and Linux Cloud-Server Hosting: http://www.gogrid.com/.
[6] Google App Engine: http://code.google.com/appengine/.
[7] Grid Computer Operating System For Web Applications—AppLogic, 3tera.: http://www.3tera.com/AppLogic/.
[8] National Institute of Standards and Technology (NIST), Cloud Standards: http://csrc.nist.gov/groups/SNS/cloud-computing/.
[9] Nimbus Science Cloud: http://workspace.globus.org/clouds/nimbus.html.
[10] Open Cloud Manifesto, Spring 2009: http://www.opencloudmanifesto.org.
[11] Open Grid Forum: Open Cloud Computing Interface (OCCI): http://forge.ogf.org/sf/projects/occi-wg.
[12] OpenNebula, the Open Source Toolkit for Cloud Computing: http://www.opennebula.org.
[13] UCI Cloud OWL Ontology: http://code.google.com/p/unifiedcloud/source/browse/trunk/ontologies/cloud.owl?r=14.
[14] Web Service Modelling Ontology (WSMO): http://www.wsmo.org.
[15] Appistry. Cloud Taxonomy: Applications, Platform, Infrastructure: http://www.appistry.com/blogs/sam/cloud-taxonomy\-applications-platform-infrastructure, 2008.
[16] R. Aversa, B. Di Martino, N. Mazzocca, and S. Venticinque. A skeleton based programming paradigm for mobile multi-agents on distributed systems and its realization within the magda mobile agents platform. *Mob. Inf. Syst.*, 4:131–146, April 2008.
[17] Steve Battle, Abraham Bernstein, Harold Boley, Benjamin Grosof, Michael Gruninger, Richard Hull, Michael Kifer, David Martin, Sheila McIlraith, Deborah McGuinness, Jianwen Su, and Said Tabet. Semantic web services framework, September 2005.
[18] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. Mcguinness, Peter F. Patel-Schneider, and Lynn A. Stein. Owl web ontology language reference. Technical report, W3C, 2004.
[19] Rajkumar Buyya, Chee S. Yeo, and Srikumar Venugopal. Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities. In *HPCC '08: Proceedings of the 2008 10th IEEE International Conference on High Performance Computing and Communications*, pages 5–13. IEEE Computer Society, September 2008.
[20] C. Hoff. Cloud Taxonomy and Ontology: http://rationalsecurity.typepad.com/blog/2009/01/cloud-computing-taxonomy-ontology.html, 2009.
[21] David Chappell. Introducing the Azure Services Platform. David Chappel and Associates Whitepaper (Sponsored by Microsoft Corporation): http://www.davidchappell.com/blog/2008/10/introducing-azure-services-platform.html, 2008.
[22] Jos De Bruijn, Holger Lausen, Reto Krummenacher, Axel Polleres, Livia Predoiu, Michael Kifer, and Dieter Fensel. The web service modeling language wsml. Technical report, DERI, October 2005.
[23] U. Villano E. P. Mancini, M. Rak. PerfCloud: GRID Services for Performance-oriented Development of Cloud Computing Applications. In *Proceedings of WETICE*. IEEE Computer Society, July 2009.
[24] Galen Gruman and Eric Knorr. What cloud computing really means. InfoWorld : http://www.infoworld.com/article/08/04/07/15FE-cloud-computing-reality1.html, 2008.
[25] Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter Patel-Schneider, and Ulrike Sattler. Owl 2: The next step for owl. *Web Semant.*, 6:309–322, November 2008.
[26] Kai Hwang. Massively distributed systems: From grids and p2p to clouds. In *Proceedings of The 3rd International Conference on Grid and Pervasive Computing—gpc-workshops*, page xxii, 2008.
[27] Jeremy Geelan. Twenty one experts define cloud computing. Virtualization : http://virtualization.sys-con.com/node/612375, 2008.
[28] P. Lairds. Cloud Computing Taxonomy. In *Procs. Interop09*, pages 201–206. IEEE Computer Society, May 2009.
[29] Alexander Lenk, Markus Klems, Jens Nimis, Stefan Tai, and Thomas Sandholm. What's inside the cloud? an architectural map of the cloud landscape. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, CLOUD '09, pages 23–31, Washington, DC, USA, 2009. IEEE Computer Society.
[30] David Martin, Massimo Paolucci, Sheila Mcilraith, Mark Burstein, Drew Mcdermott, Deborah Mcguinness, Bijan Parsia, Terry Payne, Marta Sabou, Monika Solanki, Naveen Srinivasan, and Katia Sycara. Bringing Semantics to Web Services: The OWL-S Approach. In J. Cardoso and A. Sheth, editors, *SWSWPC 2004*, volume 3387 of *LNCS*, pages 26–42. Springer, 2004.
[31] P.Kopp R.Dieckmann G.Breiter S.Pappe H.Kreger A. Arsanjani M.Behrendt, B. Glasner. Introduction and Architecture Overview, IBM Cloud Computing Reference Architecture 2.0: https://www.opengroup.org/cloudcomputing/uploads/40/23840/CCRA.IBMSubmission.02282011.doc, 2011.
[32] McGuinness, D. L., van Harmelen, F. OWL Web Ontology Language Overview. W3C Recommendation: http://www.w3.org/TR/2004/REC-owl-features-20040210/, 2004.
[33] Members of EGEE-II. An egee comparative study: Grids and clouds - evolution or revolution. Technical report, Enabling Grids for E-sciencE Project : https://edms.cern.ch/document/925013/, 2008.
[34] Dejan Milojicic. Cloud computing: Interview with russ daniels and franco travostino. *IEEE Internet Computing*, (5):7–9, 2008.
[35] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The eucalyptus open-source cloud-computing system. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '09, pages 124–131, Washington, DC, USA, 2009. IEEE Computer Society.
[36] Abhijit A. Patil, Swapna A. Oundhakar, Amit P. Sheth, and Kunal Verma. Meteor-s web service annotation framework. In *Proceedings of the 13th international conference on World Wide Web*, WWW '04, pages 553–562, New York, NY, USA, 2004. ACM.
[37] Paul McFedries. The cloud is the computer. IEEE Spectrum Online: http://www.spectrum.ieee.org/aug08/6490, 2008.
[38] R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M. Schmidt, A. Sheth, K. Verma. Web Service Semantics WSDL-S. A joint UGA-IBM Technical Note, version 1.0: http://lsdis.cs.uga.edu/projects/METEOR-S/WSDL, 2005.
[39] Bhaskar Prasad Rimal, Eunmi Choi, and Ian Lumb. A taxonomy and survey of cloud computing systems. *Networked Computing and Advanced Information Management, International Conference on*, 0:44–51, 2009.
[40] Dumitru Roman, Uwe Keller, Holger Lausen, Jos de Bruijn, Rubn Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Cristoph Bussler, and Dieter Fensel. Wsmo - web service modeling ontology. In *DERI Working Draft 14*, volume 1, pages 77–106, BG Amsterdam, 2005. Digital Enterprise Research Institute (DERI), IOS Press.
[41] Roy Bragg. Cloud computing: When computers really rule: http://www.technewsworld.com/story/63954.html, 2008.
[42] Aaron Weiss. Computing in the clouds. *netWorker*, 11:16–25, December 2007.
[43] 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.