Semantic-aware Multi-tenancy Authorization System for Cloud Architectures

Jorge Bernal Bernabe[a,b], Juan M. Marin Perez[a,b], Jose M. Alcaraz Calero[b,*], Felix J. Garcia Clemente[c], Gregorio Martinez Perez[a], Antonio F. Gomez Skarmeta[a]

[a]*Departamento de Ingenieria de la Informacion y las Comunicaciones, University of Murcia, Spain*
[b]*Cloud and Security Lab, Hewlett-Packard Laboratories, Bristol, UK*
[c]*Departamento de Ingenieria y Tecnologia de Computadores, University of Murcia, Spain*

Abstract

Cloud Computing is an emerging paradigm to offer on demand IT services to customers. The access control to resources located in the cloud is one of the critical aspects to enable business to shift into the cloud. Some recent works provide access control models suitable for the cloud, however there are important shortages that need to be addressed in this field. This work presents a step forward in the state-of-the-art of access control for cloud computing. We describe a high expressive authorization model that enables the management of advanced features such as role-based access control (RBAC), hierarchical RBAC (hRBAC), conditional RBAC (cRBAC) and hierarchical objects (HO). The access control model takes the advantage of the logic formalism provided by the Semantic Web technologies to describe both the underlying infrastructure and the authorization model, as well as the rules employed to protect the access to resources in the cloud. The access control model has been specially designed taking into account the multi-tenancy nature of this kind of environments. Moreover, a trust model that allows a fine-grained definition of what information is available for each particular tenant has been described. This enables the establishment of business alliances among cloud tenants resulting in federation and coalition agreements. The proposed model has been validated by means of a proof of concept implementation of the access control system for OpenStack with promising performance results.

Keywords: Authorization system, cloud computing, multi-tenancy, trust model, semantic web.

1. Introduction

Businesses are adapting their IT systems towards the Cloud computing paradigm where flexible and dynamic services and infrastructures are able to scale and be delivered on demand. This new paradigm enables an efficient provisioning of virtual IT architectures to third-parties, where resources are dynamically created and dismantled according to customer needs.

*Corresponding author

Email addresses: jorgebernal@um.es (Jorge Bernal Bernabe), juanmanuel@um.es (Juan M. Marin Perez), jose.alcaraz-calero@hp.com (Jose M. Alcaraz Calero), fgarcia@um.es (Felix J. Garcia Clemente), gregorio@um.es (Gregorio Martinez Perez), skarmeta@um.es (Antonio F. Gomez Skarmeta)

Cloud computing describes a logical stack divided into three different layers: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). In summary, the IaaS layer is in charge of providing the virtual infrastructure (virtual machines, volumes, networks, routing capabilities, etc.). The PaaS layer is in charge of providing middleware services which may be seen as added-value services. In turn, the SaaS layer exposes software features to be used by end-users, making use of the underlying the added-value services provided by PaaS.

However, many potential businesses which would be interested in Cloud computing are still a bit reluctant to adopt it due to security and privacy concerns. In particular, Cloud computing entails the usage of a common IT infrastructure and services which are shared between different tenants. This implies the design of strong security boundaries in order to isolate tenants when using these shared resources. Thus, the system to control the access to the available resources becomes a critical aspect in order to provide an efficient control over the usage of the cloud architecture.

Current cloud providers such as Rackspace [1] or Amazon EC2 [2] only rely on simple authentication schemes which do not provide enhanced access control capabilities beyond full access as administrator to the whole system. These authorization solutions for cloud computing (lately described in section 2) usually lack of enough expressiveness to describe advanced authorization and federation rules. The availability of advanced authorization capabilities can be a differentiating feature for cloud providers, which demand the design of suitable authorization models to enhance the access control to the cloud resources.

This paper presents an access control system suitable for cloud computing which manages grants providing high expressiveness. This enables the adoption of an advanced authorization model in which the following authorization features are supported: Role-based access control (RBAC), hierarchical RBAC (hRBAC), conditional RBAC (cRBAC) and hierarchical objects (HO). The system provides multi-tenancy support and federation capabilities allowing a fine-grained definition of what resources are available for each particular tenant. The federation capabilities are defined by means of a trust model. The trust model determines the business alliances (coalitions and federation) among cloud tenants. Although the proposed access control system is potentially suitable for all the layers of the cloud computing stack, its adaptation needs to consider some specifics of each layer, such as: i) relationships with other layers, ii) multi-vendor support, iii) information model, etc. Thus, we have decided to validate this research work over the IaaS layer. This layer constitutes the first logical step of the cloud stack and there is also a clear lack of advanced access control systems for this layer.

The remainder of this paper is organized as follows. Section 2 overviews related work about access control systems for cloud computing, indicating what are the differentiating points of our proposal. After that, section 3 describes the languages and models that are used to define the underlying system infrastructure. Then, section 4 describes the capabilities provided by our proposed authorization model. Afterwards, section 5 delves into the authorization architecture. The proposed trust model is presented in section 6. The workflow carried out during the authorization process is presented in section 7. Moreover, a proof of concept implementation and some performance results are shown in section 8. Finally, section 9 provides some concluding remarks.

[1] Rackspace available at http://www.rackspace.com/index.php

[2] Amazon EC2 available at http://aws.amazon.com/es/ec2/

2. Related Work

There is an important number of contributions in the field of access control for distributed systems. Rather than providing a complete historical review, this section is focused only in those which provide multi-tenancy support, which is a required feature in order to fit cloud computing. In essence, multi-tenancy is the ability to efficiently deal with multiple administrative domains (tenants) that are using the same service which, in turn, has isolated resources belonging to particular tenants. Many distributed systems like Grid and Cloud computing demand multi-tenancy support. Cloud computing is usually related to a single provider and homogeneous information models whereas Grid computing is designed on the assumption of multiple providers with heterogeneous information models [1]. Thus, although there are several access control systems provided for grid computing, they do not directly fit in cloud computing.

We proposed a semantic-aware access control system for grid computing that enables basic coalitions and federation capabilities [2]. This work assumes heterogeneous tenants and enables basic federation capabilities between them. This is achieved by means of the virtual organization concept, which is articulated by the alignment of information available for tenants as homogenization tool. In case the reader is interested, Marin et al. [2] provide a comprehensible overview on grid-related authorization systems. However, they do not fit cloud computing for the same reason, i.e. they are based on design principles different from cloud computing characteristics.

Marin et al. [2] is, in turn, the application to Grid computing of a previous contribution in which we designed an authorization framework for distributed environments providing a multi-tenancy authorization model with support for RBAC, hRBAC, cRBAC and HO [3]. This access control model is the basis of the work available in both Marin et al. [2] and this contribution. Although there are some similarities, there is a significant work done in order to achieve an access control model which really fits cloud computing. In SECRYPT 2011 [4], we presented a short paper in which we focus on describing an overview of an access control system based on the design assumptions of cloud computing. It is mainly focused on providing the description of a new information model to cope with cloud-related concepts like *Virtual Machine* or *Hosted Operating System*. Now, we are significantly extending that contribution, mainly by providing the following features: Firstly, a real implementation of the access control system which has been integrated in the well-known Open Stack cloud platform. Secondly, a completely new trust management model in order to provide fine-grained cloud federation capabilities. Finally, a complete performance evaluation of the proposed prototype is also provided.

Regarding access control models for Cloud computing, Li et al. [5] provide a basic multi-tenancy access control model with discretional access control (DAC) support. Li et al. [6] extends this model providing support for role management (RBAC) for cloud computing. Shirisha and Geetha [7] provide the next step supporting role hierarchies (hRBAC) in an access model designed to control the invocation of methods available in cloud computing APIs. Tsai and Shao [8] provide a semantic-aware multi-tenancy access control model with hRBAC and cRBAC support. Authors use an ontology for building up the role hierarchy for a specific domain. Ontology transformation operations algorithms are provided to compare the similarity of different ontologies.

Pereira [9] and Xu et al. [10] provide access control systems for the Cloud with analogous functionality. The main difference is that Pereira is focused on the IaaS layer whereas Xu et al

is focused on the SaaS layer. They provide not only a multi-tenancy access control model with hRBAC and cRBAC support, but also a dynamic activation of roles in order to control a proper separation of duties in the Cloud. Danwei et al. [11] delves into an access control system based on the *Usage CONtrol* access model (UCON) [12] which includes negotiation techniques in order to provide federation capabilities in Cloud computing. UCON model encompasses hRBAC, cRBAC, and attribute-based access control (ABAC) support. Fall et al. [13] provide a similar access control system for Cloud computing with the differentiating feature that federations between tenants are dynamically stablished, according to a risk management model in charge of deciding if two tenants can collaborate according to their previous interactions. Calero et al. [14] has provided an advanced multi-tenancy authorization model with RBAC, hRBAC, HO support for Cloud computing based on authorization statements defined by means of paths. This approach provides efficiency and performance making the system scalable. However, path-based representation could suffer of expressiveness limitations when authorization information needs to be expressed over information models that cannot be expressed using paths.

All the previously described models represent good attempts to provide access control models adapted to cloud computing. However, there is still an important effort in the way of providing efficient and highly expressive models. These models have been compared with respect to our contribution presented in this paper in order to provide a clear overview about what is our main contribution to the field.

1	RBAC	Role-Based Access Control. It allows to assign users to roles and define authorization statements over roles.
2	hRBAC	Hierarchical RBAC. It allows to manage sub-roles (children) which inherits the privileges of the parent roles.
3	PBAC	Conditional RBAC (cRBAC) or Policy-based Access Control (PBAC). It allows to dynamically assign privileges only when some conditions are fulfilled. This conditions are defined with information of the protected resources.
4	HO	Hierarchical Objects. It allows to create object hierarchies and inherit the privileges defined for the object on top of the hierarchy to its children.
5	SoD	Separation of Duties. It allows to define mutually exclusive role assignations to subjects.
6	CBAC	Context-based Access Control. It allows to dynamically asign privileges only when some conditions are fulfilled. This conditions are defined with contextual information of the system, for example, time.
7	ABAC	Attribute-based Access Control. It allows to define authorization statements for attributes and when these attributes are assigned to users, they receive the specified privileges.
8	SF	Simple Federation. It allows to share authorization information between tenants in order to enable a tenant to define authorization statements using the information of another tenant. It is simple because a tenant A trusts or does not trust another tenant B and it implies that A enables B to access all the information model of A.
9	MT	Multi-tenancy. It allows to manage different administrative domains using the same shared infrastructure.
10	FGF	Fine-gain Federation. it allows a selective choice of the information that a tenant A wants to share with another tenant B (rather than sharing the complete information as in SF).
11	CD	Conflict Detection. It allows to detect inconsistencies in the authorization information in order to validate the information.
12	CCI	Cloud Computing Integration. It determines if the proposed model has been designed to fit Cloud computing.
13	US	Unified Semantics. A common problem in authorization systems is that there is usually a mismatch between the semantics used to define the authorization model and the semantics used to define the information model. That can potentially cause security holes. An homogenization of such semantics bring an added-value to the proposal.
14	VS	Virtualization Support. It determines whether the proposed model has direct support to define the information model needed in a virtualized Cloud computing environment.
15	PV	Performance Validation. It indicates if the research work have empirical results or not.
16	AMC	Authorization Model Comparison. It indicates whether the research work has a comparative analysis between the different authorization models which are supported.

Table 1: Features compared across access control models

The aspects to be compared are described in table 1 and the comparison is shown in table 2. Note that the columns of table 2 matches with the IDs specified in table 1. Note that Marin et al. [2] provides a fine-grained federation model. However it is labelled as partial because it does not fit cloud computing requirements at all.

As can be seen in table 2, The proposed access control model combines the previous proposals in order to go a step forward by describing a multi-tenancy authorization model suitable for Cloud computing, including several advantages with respect to its predecessors. The model provides a high level of expressiveness, enabling advanced authorization features such as cRBAC, hRBAC,

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Li et al. [5]									X			X				
Li et al. [6]	X								X			X				
Sirisha and Geetha [7]	X	X							X			X				
Tsai and Shao [8]	X	X	X			X		X	X			X	X	X		
Pereira [9]	X	X	X		X	X		X	X			X				
Xu et al. [10]	X	X	X		X	X			X			X				
Danwei et al. [11]	X	X	X		X	X	X	X	X			X				
Fall et al. [13]	X		X		X	X	X	X	X			X				
Calero et al. [14]	X	X		X				X	X			X		X	X	
Alcaraz et al. [3]	X	X	X	X	X	X	X	X	X		X		X	X	X	
Perez et al. [2]	X	X	X	X	X	X	X	X	X	*partially	X		X		X	
Bernabe et al. [4]	X	X	X	X		X	X	X	X			X	X	X		
Our proposal	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Table 2: Comparison of the Different Access Control Models Analyzed

GTRBAC and hierarchical objects, as well as a fine-grained control over the definition of federations and coalitions between tenants. Moreover, it has been integrated in a real Cloud computing system and its performance has been analyzed.

3. Information model

The proposed access control system uses a model to represent the resources that are managed and protected by the system. This model is represented by means of an ontology defined in the Ontology Web Language 2 (OWL 2) [15] and the Semantic Web Rule Language (SWRL) [16]. These languages are based on formal methods and they constitute a remarkable added value since they enable the usage of reasoners to infer new knowledge. The reasoner is able to derive additional information not explicitly specified in the ontology, and to perform a formal validation and verification of the model. The Description Logics formalism in which these languages are based is kept within the decidability bounds, so that inference processes are performed in a finite time.

Several standard models like the Common Information Model (CIM) [17] and the Open Information Model (OIM) [18] have been designed to model information systems. CIM, which was created by the Distributed Management Task Force (DMTF), has been selected as base model for our access control system due to its advantages. Namely, wide coverage of information systems, extensibility mechanisms, independency of the representation language, and widely used in several research works like [19] and [20], as well as in a variety of large systems such as SAP, Microsoft Windows and VMWare, among others. The representation of CIM into OWL results in a semantically enriched information model.

For the shake of simplicity, only managed resources at the IaaS layer are exposed in this section but the model is easily extensible to provide support for the PaaS and SaaS layers. Common concepts managed in cloud infrastructures include virtual machines (VMs), virtual networks which interconnect the VMs and volumes attached to the VMs. In conclusion, the model should contain enough concepts and relationships to define the different elements supported by the virtual infrastructure. Figure 1 shows a UML representation of the main CIM classes employed in our information model.

Among others, `VirutalComputerSystem` is an important class which allows to manage the virtual machines of the cloud. Thanks to the `installedOS` and `runningOs` associations, the model is able to deal with the operating system of each virtual machine. `VirtualSystemSettingData` defines the aspects of a virtual system through a set of virtualization specific properties (e.g.

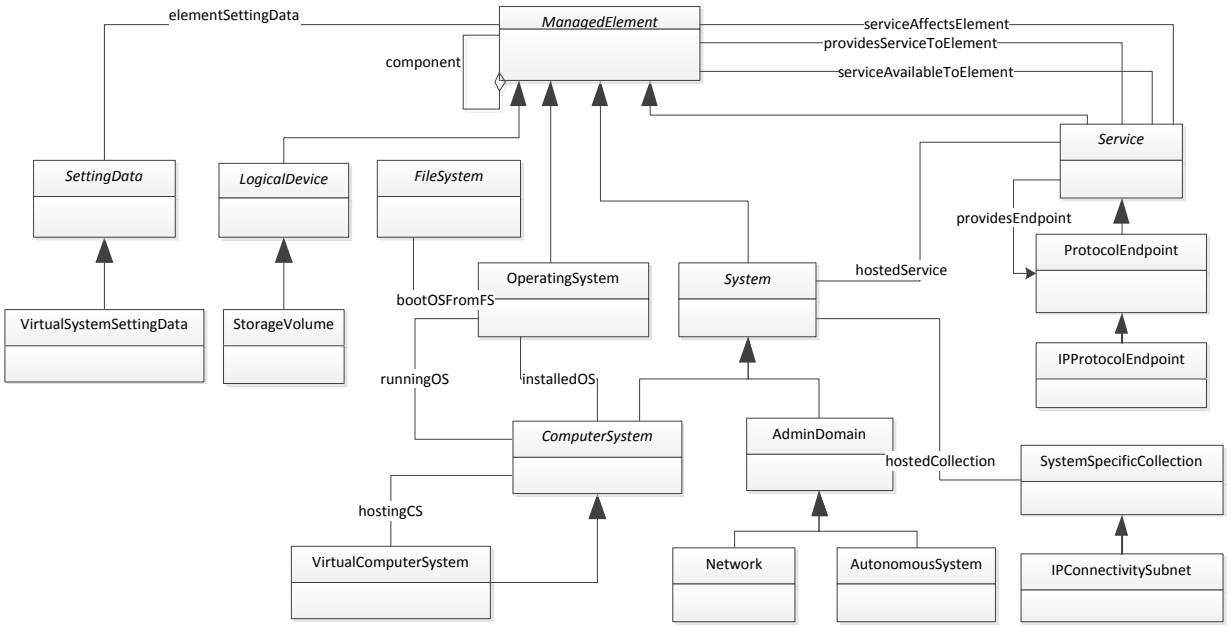


Figure 1: UML representation of the information model

SnapshotDataRoot, ConfigurationFile, AutomaticStartupAction, VirtualSystemType, SwapFile-DataRoot...).

Each cloud tenant in our model is represented by an `AdminDomain`. An `AutonomousSystem` is used to segregate the system by routing into a set of separately administered domains, having each one its own independent set of policies. `LogicalDevice` is an abstraction or emulation of a hardware entity which enables the management of devices at the IaaS layer. We use the `StorageVolume` class, which in turn extends `LogicalDevice`, for the representation and management of volumes in the Cloud.

4. Authorization Model

The access control system must determine whether a given subject is granted with a privilege over a given resource or not. The authorization decisions are based on the definition of authorization statements. An authorization statement is defined by a 3-tuple $(Subject, Privilege, Resource)$ which establishes that a subject has a privilege on a resource. When a user attempts to access a Cloud resource, the access control system seeks along the available authorization statements for the requested user (`Identity`), action (`Privilege`) and resource (`ManagedElement`). By default, if there is no matching authorization statement in the model, the access to the resource is denied. These authorization statements can be represented in the ontological authorization model by means of the set of CIM concepts and relationships depicted in Figure 2.

An `Identity` represents a user that can be validated during the authentication stage, whereas a `Role` represents a set of responsibilities within an organization. Identities can be assigned to a `Role`. `Privilege` is the base concept for all types of activities/actions which are granted or denied

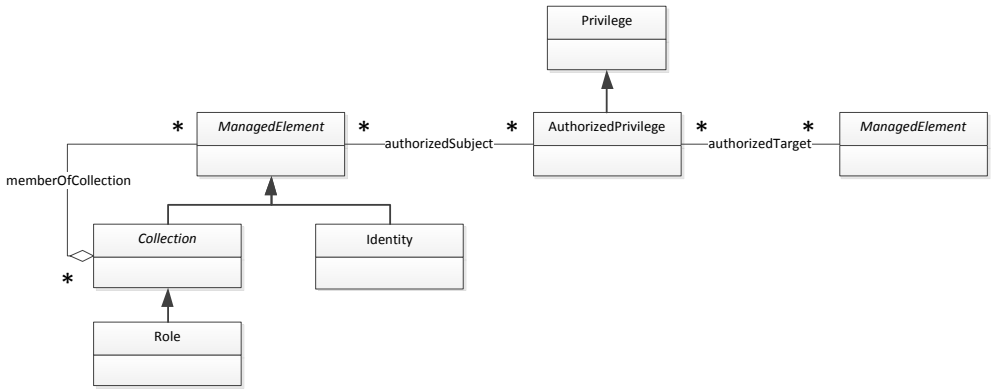


Figure 2: UML diagram of authorization concepts

to a subject for a given resource. `AuthorizedPrivilege` extends `Privilege` to describe current privileges in the authorization model. In order to increase the readability, `AuthorizedPrivilege` class will be henceforth referred to as `Privilege`, indistinctly.

With the aim of clarifying the authorization capabilities supported by the access control system, Figure 3 depicts an example where different cloud tenants share different resources. In this example, a user of tenant A (belonging to the role Lab Head) tries to access resources (in this case to the VMs) of tenant B. Likewise, a Lab Head user of tenant B tries to access to the network of tenant A.

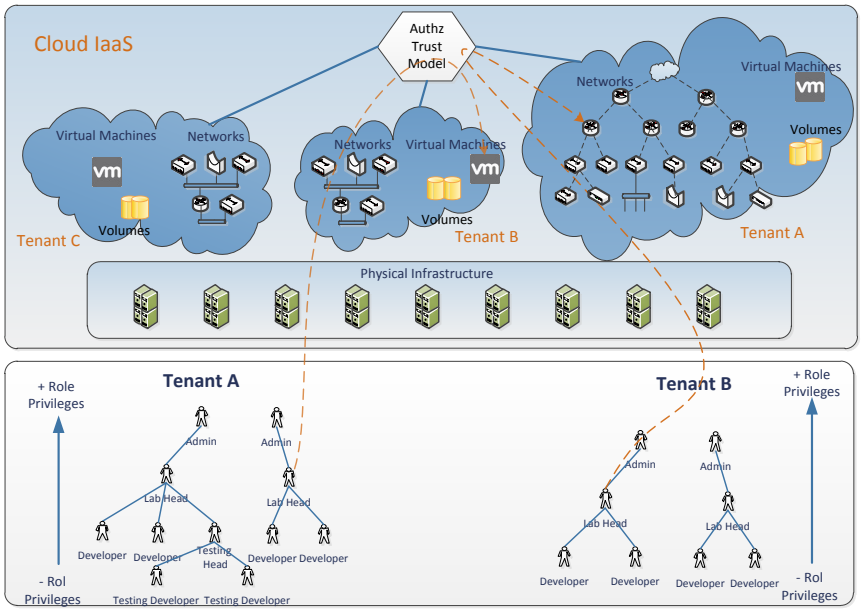


Figure 3: Cloud authorization example

To support Role-based Access Control (RBAC) in the authorization model, the *Subject* defined in the 3-tuple can be either represented by an `Identity` or a `Role`. In Figure 3, tenant A

describes different roles such as Admin, Lab Head and Developer in order to assign later different privileges to the role as a whole. Role-based Access Control (RBAC) capability is supported in the authorization model by means of a rule (expressed in SWRL language) which propagates the privileges assigned to a role to the identities belonging to that role. This rule is depicted in Rule 1.

$$Role(?r) \wedge AuthorizedPrivilege(?p) \wedge authorizedSubject(?p,?r) \wedge \quad (1)$$
$$Identity(?i) \wedge memberOfCollection(?i,?r) \quad (2)$$
$$\rightarrow \quad (3)$$
$$authorizedSubject(?p,?i) \quad (4)$$

Rule 1: Role privileges propagation meta-rule

The rule selects any role having some associated privileges. It also selects any identity which is member of that role (line 2). As a result, any privileges granted to the role is also granted to the identities belonging to such role. Notice the power of using this ontological approach for the authorization model, since just one rule is enough to provide RBAC capabilities to the whole access control system.

Multi-tenancy is an important aspect which characterizes a cloud environment. It implies a new extension to the authorization statements, adding the concept of Issuer. This concept is introduced to represent the tenant who defines the statement. It is required in order to recognize the owner of the statement and therefore to limit the information model that is taken to perform the authorization decision. Thus, authorization statements are now represented by the 4-tuple $(Issuer, Subject, Privilege, Resource)$. This tuple can be read as *'The tenant* Issuer *states that the user or role* Subject *has the permissions defined by* Privilege *over the element* Resource*'*. The *Issuer* is represented using the CIM `AdminDomain` class in the information model.

Conditional RBAC (or Generalized RBAC) is an advanced capability rarely supported by current cloud access control systems. This feature supports the assignment of privileges to environment roles. These environment roles are activated according to the changes specified in environmental conditions defined by the administrators. Thus, the permissions defined in the authorization statements only apply when the specified conditions are fulfilled, which usually refer to context information of any nature. It requires the extension of the authorization statements to a 5-tuple $(Issuer, Subject, Privilege, Resource, Conditions)$. Note that conditional RBAC includes many previous research work in authorization models such as attribute-based accessing control (aBAC) - using user attributes as conditions - or Generalized spatial time RBAC (GST-RBAC) - using temporal and spatial context information as conditional -, including all the proposals in which authorization statements are granted when certain conditions are fulfilled.

The usage of SWRL to represent the authorization statements enables to cope with this capability in the access control system. The antecedent of the SWRL rule represent the *Conditions* of the tuple. It includes references to elements of the information model as conditions. This means that only when the conditions are fulfilled, the consequent is triggered granting the privileges. When the administrator does not provide *Conditions* in the authorization statement, it means that

the rule will have an empty antecedent. Thus, the rule will be always triggered since no conditions have been specified and the permissions defined by the statement always apply. This is the way in which the authorization statements are stored in the model.

The enablement and disablement of roles is achieved by adding and removing identities associated to the Role. This approach allows that two users with the same role can keep it activated or deactivated in different periods of time. For instance, a given user can keep the *Admin* role activated only from Monday to Friday, while another user can keep it activated on weekends. If the role needs to be completely deactivated, it is just required to remove all the users associated to the Role. The antecedent of a SWRL rule describes the activation conditions and the consequent creates or removes the *memberOfCollection* association between *Subjects* and *Roles* to activate or deactivate them.

In the example represented in Figure 3, a user *Bob* acting as *Developer* of tenant A tries to perform an action on one of the virtual machines of his own organization. Listing 1 shows a fragment of the model managed by the access control system using RDF/XML [21] syntax, which is the primary exchange format for OWL 2 ontologies.

```
<rdf:RDF>
  ...
  <Identity rdf:about="#Bob">
    <instanceID>Bob</instanceID>
    ...
  </Identity>
  <Role rdf:about="#Developer">
       <instanceID>Developer</instanceID>
       ...
  <Role>
  <memberOfCollection rdf:about="#BobDeveloper">
   <collection>Developer</collection>
   <member>Bob</member>
  </memberOfCollection>
  <VirtualComputerSystem rdf:about="#VM1">
    <name>VM1</name>
    <operatingStatus>In Service</operatingStatus>
    ...
  </VirtualComputerSystem>
</rdf:RDF>
```

Listing 1: Sample model fragment

In this context, the administrator may decide the following authorization statement: *"Developers are allowed to stop VM1 only if that machine is not performing a snapshot"*. This statement can be represented by the 5-tuple $(CompanyX, Developer, Stop, VM1, VM1.OperatingStatus \neq Snapshotting)$. Rule 2 shows the representation of this authorization rule in SWRL abstract syntax.

The first two lines of Rule 2 specify the corresponding virtual machine (*VM1*) and check the condition that it is not performing a snapshot. Line 3 selects the subject of the authorization statement, i.e. the role *Developer*. The consequent of the rule establishes the corresponding privilege in terms of user concepts and relationships.

Since the status of the virtual machine *VM1* described in Listing 1 is "In Service" - which is different from "Snapshotting" - the conditions of the statement are fulfilled. Thus, Rule 2 grants the stop privilege to the role *Developer*.

$$VirtualComputerSystem(?vm) \wedge name(?vm,?n) \wedge equal(?n,'VM1') \wedge \quad (1)$$
$$operatingStatus(?vm,?st) \wedge notEqual(?st,'Snapshotting') \wedge \quad (2)$$
$$Role(?r) \wedge instanceID(?r,?id) \wedge equal(?id,'Developer') \quad (3)$$
$$\rightarrow \quad (4)$$
$$AuthorizedPrivilege(?p) \wedge activities(?p,'Stop') \wedge \quad (5)$$
$$auhtorizedSubject(?p,?r) \wedge auhtorizedTarget(?p,?vm) \quad (6)$$
$$(7)$$

Rule 2: Sample authorization statement rule

Thanks to the usage of Rule 1, the access control system also infers the authorization fragment which allows *Bob* - as identity belonging to the *Developer* role - to access the virtual machine. The inferred fragment of the model is depicted in Listing 2.

```
<rdf:RDF>
  ...
  <AuthorizedPrivilege rdf:about="#StopPriv">
    <instanceID>example:stopPriv</instanceID>
    <activities>Stop</activities>
    ...
  </AuthorizedPrivilege>
  <auhtorizedSubject rdf:about="#DevelopersStop">
    <privilege rdf:resource="#StopPrivilege"/>
    <privilegedElement rdf:resource="#Developers"/>
  </authorizedSubject>
  <authorizedTarget rdf:about="#StopVM1">
    <privilege rdf:resource="#StopPrivilege"/>
    <targetElement rdf:resource="#VM1"/>
  </authorizedTarget>
       ...
       <auhtorizedSubject rdf:about="#Bob">
    <privilege rdf:resource="#StopPrivilege"/>
    <privilegedElement rdf:resource="#Bob"/>
 </authorizedSubject>
</rdf:RDF>
```

Listing 2: Inferred information example of an authorization rule

Hierarchical Role-based Access Control (hRBAC) extends RBAC with the aim of defining role hierarchies. These hierarchies establish privilege inheritance between roles, making a child role to inherit all the privileges defined for parent roles in the hierarchy. The major motivation for adding role hierarchy to RBAC is to simplify role management. A child role A can be defined as member of another parent role B. Privilege inheritance is achieved by the definition in OWL of the `memberOfCollection` property as transitive. It makes the system to consider instances belonging to any parent role, to be also belonging to its child roles in the hierarchy. In the example depicted in Figure 3, this feature allows any new privilege assigned to a role, for instance *Testing Developer*, to be automatically propagated to the upper role and so on. Thus the privilege is automatically assigned to the *Testing Head* role, the *Lab Head* role and the *Admin* role, respectively.

Hierarchical objects (HO) capability allows the privileges applied to a given object to be also applied to all its children objects in the hierarchy. This enables to achieve a high-level management

of the access control system. In the example shown in Figure 3, the access control to the virtual network of tenant A can be effectively managed using this capability. A network represents an inherent hierarchical structure. In our domain model, different levels of networks and subnetworks can be defined as parents or children. So that, privileges affecting to a given network can be applied to its subnetworks and even to the virtual resources located in such subnetworks. The CIM allows the representation of parent-children relationships between resources by means of the `Component` association. This association is defined in the model as transitive, enabling the authorization system to recognize all subcomponents and dependent objects of another object through the whole hierarchy. A SWRL rule can be easily established to propagate the privileges throughout these objects.

Separation of duties (SoD) enables the protection of the fraud that might be caused by users. Since it is possible that two given privileges should not be associated to the same role at the same time and in the same place, the Static Separation of Duty feature is quite important in any access control system. This situation can be detected in our access control system by looking for a provoked inconsistency in the knowledge Base (KB).

The reasoning capabilities available in the Semantic Web enable advanced features like conflict detection and even conflict resolution. The usage of OWL and SWRL as modeling languages for information systems provides different ways to detect conflicts that may appear in the KB. A customized set of SWRL rules can be used to include conflictive privileges in the antecedents and force a conflict by including contradictory facts in the KB. Since our models are described in OWL, a reasoner can be used to detect an inconsistence in the KB. When two contradictory facts are hold on the KB, it becomes inconsistent and reasoners are able to detect this situation. Thus, the consistency checking process of DL reasoners can be used to detect semantic conflicts. For more detailed information regarding semantic conflict detection refer to [22].

4.1. Authorization Models comparison

The previous section has defined different authorization capabilities that are supported by the access control system. This subsection provides some experimental results to evaluate and compare the authorization models which result from the application of the capabilities explained above, i.e. DAC, RBAC, hBAC and HO. These authorization capabilities can be compared since all of them can represent the same scenario and system behavior. However, conditional RBAC (cRBAC) can not be analyzed because the absence of that capability in the system can not be replaced by static definitions. That is, it affects not only to the ontology system definition but rather to the behavior of the authorization system.

As explained before, the authorization capabilities are implemented by means of SWRL rules. Since the time required to evaluate the SWRL rules varies according to the nature and number of the statements, the results depend on the selected scenario. Thus, the obtained results can change substantially according to different factors. Among them, the depth in the hierarchical system structure (e.g. Subnets and AdminDomains), the number of Roles and its levels of hierarchical depth or the number of Identities and privileges defined.

Nevertheless, describing a generic scenario can be useful to figure out the time required by the system to evaluate the rules, and therefore to support certain authorization capabilities. The chosen reference scenario is composed of 100 Identities, 21 Roles, 30 Authorization Privileges and 200

Managed Elements. Roles are arranged in a three tier fashion where 3 of them are declared on top of the hierarchy, 6 in the medium level and 12 in the bottom. The Managed elements are composed of 6 AdminDomains, 24 Subnets associated to them and 170 virtual machines associate to these subnets. The 100 identities are assigned to the different Roles. Finally, authorization statements are defined to grant the privileges to both Roles and Identities.

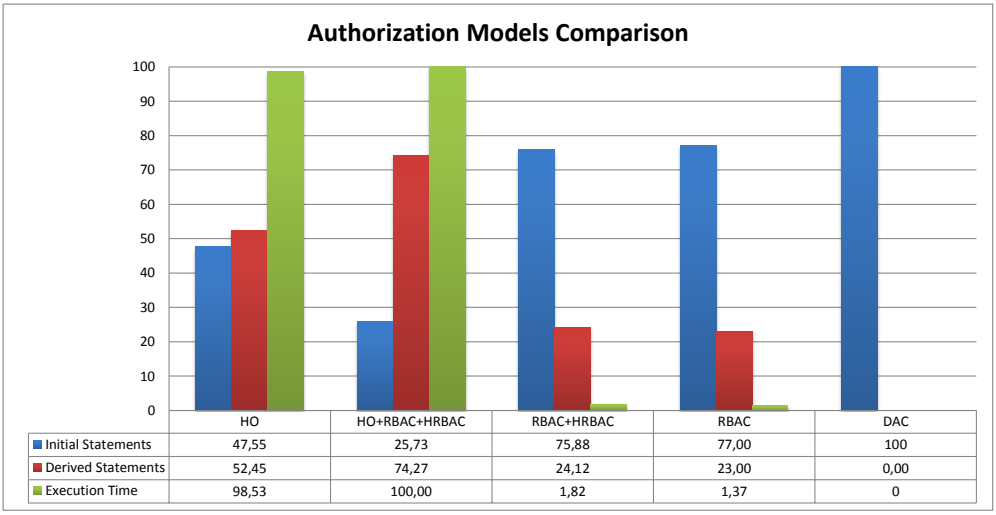
Privileges are assigned to the upper level in the system hierarchy i.e. to AdminDomains. Thus, after executing the HO SWRL rule, all the identities obtain the grants to access to the virtual machines defined in the bottom of the hierarchy. If hRBAC and RBAC are used, privileges assigned to the upper level Roles are inherited by the identities in the bottom of the hierarchy.

Five different configurations with five different populations have been made up to compare the authorization models. All the populations represent the same described scenario. Differences lie in the amount of rules as well as the ontology model statements hold at each one. Although the amount of statements at each population is different, executing the SWRL rules lead populations to be equivalent to each other. This is due to the fact that SWRL rules derive the equivalent statements as if they were created beforehand in the ontology model.

The first population is envisaged to evaluate HO in an isolated way. This population initially includes all the required model statements of the scenario as if they were generated by rules of hRBAC and RBAC. The second population is used to evaluate HO, hRBAC and RBAC authorization capabilities all together. This population requires a minimum initial amount of statements since many of them are derived by the rules. The third population is used to evaluate the behavior of hRBAC and RBAC. Notice that statements that may be generated by the HO rule are included beforehand in the population to make it comparable. The fourth population is used to evaluate RBAC independently. Similarly, the statements generated by the SWRL rules of hRBAC and HO are previously included in the population. The fifth population represents a reference population without any rule, where all the statements have been previously generated. This population is equivalent to having a Discretionary Access Control (DAC) system without hRBAC, RBAC nor HO.

Figure 4 shows the achieved results. There are three different data series in the graphic. *Initial Statements* represents the percentage of statements, with respect to the total, that are required to evaluate the authorization capabilities depicted in the x-axis. The total amount of statement in the scenario is 8600. This number coincides with the amount of *Initial Statements* in the DAC configuration, where everything is statically defined in the model and nothing is derived by rules. *Derived statements* represents the amount of statements which are inferred by SWRL rules. It should be noticed that *Initial Statements* plus *Derived statements* always sum up to the 100% of the statements, regardless the authorization configuration.

The configuration (HO + hRBAC + RBAC) provides the authorization system with the maximum level of expressiveness, so that it requires the highest computational time to be enforced. For this reason, this *Execution time* has been taken as reference (100% value in the chart) to be comparable with the rest of configurations. As can be seen from the figure, the evaluation of HO alone, requires a lot of time to be completed when compared with RBAC or HRBAC. It can be explained because it is a complex SWRL rule that derivates a lot of statements, more than 50% of the total. In DAC, since there is no SWRL rule, there is nothing to evaluate, and the amount of derived statements is 0.



	HO	HO+RBAC+HRBAC	RBAC+HRBAC	RBAC	DAC
Initial Statements	47,55	25,73	75,88	77,00	100
Derived Statements	52,45	74,27	24,12	23,00	0,00
Execution Time	98,53	100,00	1,82	1,37	0

Figure 4: Authorization Models Comparison

5. Architecture

The proposed architecture is illustrated in Figure 5. It has been designed to be placed at the IaaS layer of the cloud stack. It takes authorization decisions for cloud resource access attempts according to the authorization statements described by both tenants and cloud providers.

The `IaaS API` is the API already provided by all current IaaS cloud solutions like Eucalyptus, OpenStack, Amazon EC2 or RackSpace, among others. The `Interceptor` module captures the requests coming from users through the IaaS API in order to redirect them to the access control system. The `Interceptor` component is also responsible for returning the request back to the IaaS API, according to authorization decision carried out by the access control system.

The access control system is implemented by means of the `SemanticAuthzService`. The main aim of this component is to evaluate whether user requests have enough grants to perform the requested actions on the resources. It is composed of two main different submodules: the `Authorization Engine` and the `Trust Manager`. The `Authorization Engine` performs authorization reasoning based on the SWRL rules and the OWL ontology. In turn, the `Trust Manager` manages the privacy of tenants information according to established trust relationships.

The `Authorization API` component extends the IaaS API by enabling the description and insertion of authorization statements in the system. Such an API translates the statements to authorization rules in SWRL format in order to be later stored in the `Knowledge Base`. This component keeps the authorization information needed to carry out the authorization decisions. It includes both the authorization rules and the information model instances that represent the virtual infrastructure being managed.

Note that customers using the `Authorization API` do not require to understand neither OWL nor SWRL. The API helps to deal with rule definition, abstracting users from these languages. The concepts of the ontology are presented to the user so that he is able to define the conditions of the rules based on these concepts.

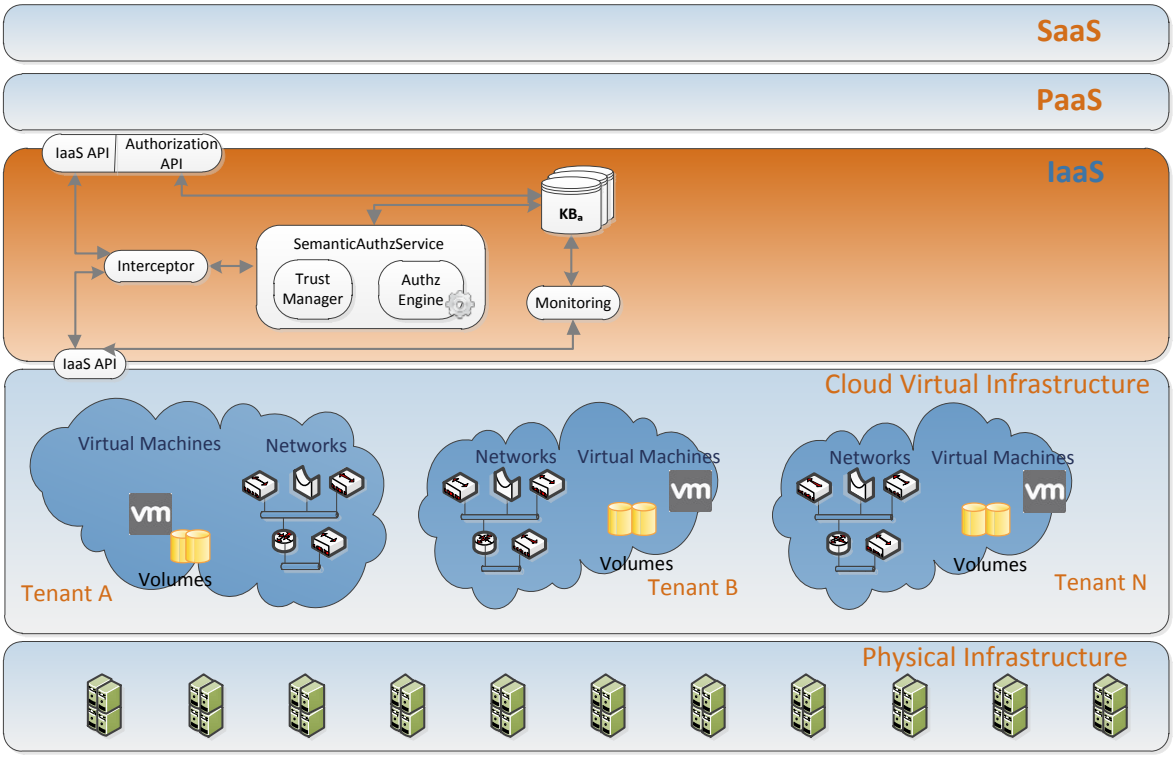


Figure 5: Semantic Authorization Architecture for cloud.

The access control system requires being up to date with the events that take place in the virtual infrastructure in order to achieve proper authorization decisions. Thus, the `Monitoring` component obtains information about the underline virtual environment and updates the KB accordingly. This update is done thanks to a subscription mechanism enforced in the cloud IaaS API. The module translates the source event messages into our ontological information model representation which is maintained in the KB.

6. Trust Management

A collaboration agreement between two tenants can be represented by means of a trust relationship. Trust relationships define the level of security and privacy that should be ensured between tenants. Our solution provides a fine grained trust model where tenants can describe which specific information from its information model is shared with other tenants. If a tenant A trusts another tenant B - denoted as $A \prec B$ -, it implies that B has available certain information of A to describe its authorization rules. Thus, a trust relationship can be seen as a 3-tuple ($Trustor$, $Trustee$, $ContextInfo$) describing that a $Trustor$ trusts a $Trustee$ by exposing the $ContextInfo$ information. Then, the trustee is able to use that context information belonging to the trustor in

order define its authorization statements. This mechanism enables tenants to control the access to their the resources from other tenants.

A common situation is a tenant B allowing another tenant A to use its subjects (indicating the subjects in the $ContextInfo$). It would permit A to describe authorization rules taking into account subjects of B. This approach enables A to grant subjects of B to access to A's resources.

It could also happen that the trust relationship $A \prec B$ implies that B allows A to use objects and resources (not only subjects) in A's authorization rules as context information, i.e. as conditions. To do so, B only needs to specify that information in the $ContextInfo$ of the trust relationship. As an example of this situation, a tenant A could describe an authorization rule stating that a user of tenant B could access some of A's resources only if B's virtual machines of a given network are not available. It is important to remark that this feature does not imply that A can grant privileges for his users to access B's resources. This can only be specified by B. In other words, a tenant is the only one which can determine its permissions over its resources.

Trust relationships are not transitive, i.e. if $A \prec B$ and $B \prec C$, it does not imply that $A \prec C$. We decided to not automatically manage such a feature in order to keep the federation model simple. Obviously, this is supported just by explicitly inserting such trust relationships. Moreover, trust relationships are not symmetric, i.e. if $A \prec B$, it does not imply that $B \prec A$. This enables more control over the definition of federation scenarios. Analogously, in case this feature is needed, the inclusion of the corresponding symmetric trust relationship is enough to support it into the system. Finally, by default nobody trusts anyone else unless there is an explicit statement for this.

The trust information is used by the `TrustManager` module available in the architecture. Upon a request, this module is in charge of providing only the knowledge of those organizations which trust each other to take the authorization decision. It controls the privacy of all the information stored in the `Knowledge Base`. The module selects the information to be used to make the authorization decision taking into account the issuer requesting authorization in order to provide multi-tenancy in the cloud environment.

Our architecture keeps a different KB for each tenant. Each of these KBs contains the information model and the authorization statements for a particular tenant. Each KB also keeps the knowledge allowed by its trusted tenants according to the trust models defined with the aforementioned 3-tuple. This approach is safer and more secure than having just one KB for all the information since it allows to physically isolate sensitive information from different tenants.

The `TrustManager` module also manages the life cycle of the different KBs keeping them up-to-date according to the changes in the trust relationships between tenants. This update implies removing knowledge from the KB when a tenant cancels a trust relationship or updating the KB when a new trust relationship is established. Holding different KBs with information for different tenants implies that the `Monitoring` component also has to be aware of these relationships in order to update the KBs according to the information coming from the virtual infrastructure.

7. Authorization Process

Considering a symmetric trust relationship between two tenants A and B defined by $A \prec B$ and $B \prec A$, when a user of A tries to access to a resource belonging to B, the request first reaches the `IaaS API`. Then, the `Interceptor` takes the incoming request and forwards it to the

SemanticAuthzService. The Trust Manager chooses the proper KB according to the organization to which the request resource belongs. When the Authorization Engine is aware of the KB which should be used, it reasons using the information model and authorization statements and derives the authorization decision. Finally, the authorization response is sent back to the Interceptor which forwards it to the IaaS API, granting or denying the access to the resource.

The AuthzEngine makes authorization decisions based on the reasoning process performed by an OWL and SWRL reasoner. The underlying information model and the authorization statements are specified as SWRL rules and an OWL ontology stored in the KB. The reasoner uses this KB together with the semantic rules and the authorization model information to carry out the reasoning process.

Three main operations are available in the reasoner. Firstly, the reasoner performs an **Inference** process that allows new knowledge to be derived using the information available in the ontology. Note that the SWRL rules which compose the authorization model derive new information that is used to make the authorization decision. An example of inferred knowledge is shown in Listing 2. Secondly, the reasoner also performs a **Validation** process to detect possible violations of the constraints expressed in the ontology. It mainly consists of a global checking across the schema and the current instances looking for inconsistencies. Thirdly, the reasoner allows to **Query the ontology**, what includes instance recognition and inheritance recognition. The former consists in testing if an individual is an instance of a class and the latter if a class is a subclass of another class - or if a property is a sub-property of another one. This enables the formulation of generic queries referring to abstract concepts, being the system able to recognize instances belonging to concrete subclasses or sub-properties of such a concept.

To make the authorization decision, the SemanticAuthzEngine queries the reasoner to look for OWL instances in the KB which compose the authorization statement. Listing 3 shows the authorization query in SPARQL that examines the KB looking for a privilege that allows an identity to perform a certain action on a given resource.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?privilege
WHERE {
      ?privilege rdf:type AuthorizedPrivilege .
      ?privilege AuthorizedPrivilege_activity #action# .
      ?identity rdf:type Identity
      ?identity Identity_name #identity# .
      ?privilege authorizedSubject ?identity .
      ?privilege authorizedTarget ?target .
      ?target ManagedElement_ID #target# .
     }
```

Listing 3: SPARQL authorization query

Listing 3 searches an AuthorizedPrivilege instance which is associated with the given identity, target and action. It can be applied to the example of inferred knowledge provided in rule 2. The query has three input parameters: the identity that is performing the request, which in the example is *Bob*; the target that is being accessed, which in this case is the virtual machine *VM1*; and the action performed against the target, which in this case is a *Stop* operation over the virtual machine *VM1*. In case the query does not return any privilege, the AuthzEngine denies the identity to perform the requested action over the resource.

8. Implementation and Performance

A proof of concept prototype of the `SemanticAuthzService` has been developed (see figure 5). It has been integrated into an OpenStack [23] IaaS cloud provider. OpenStack is an open initiative to provide an IaaS stack for cloud computing. It is becoming one of the most widely used IaaS stacks solutions, with an important and increasing list of companies and organizations involved, such as Rackspace, Nasa, Citrix, AMD, Intel, NEC or Hewlett Packard, to name a few. This is the main reason for which we have chosen OpenStack for our research work.

OpenStack provides three main components to enable an IaaS platform. Namely, the OpenStack Compute (Nova), the OpenStack Image Service (Glance) and the OpenStack Object Storage (Swift). Nova is the cloud fabric controller, in charge of controlling virtual machine instances and networking. Glance can be seen as an image repository which enables lookup and retrieval of virtual machine images. Swift provides a high capacity, scalable and reliable storage system for the cloud infrastructure. It can also be used by Glance as storage service.

Currently, some of these Open Stack components have their own authentication and authorization mechanisms (e.g. swauth for Swift). Leveraging the access control to the IaaS in several different modules increases the complexity of authorization management. This may result in inconsistencies, conflicts or even security flaws. The implementation developed in our proposal is offered to the rest of system components as an autonomous service. This approach enables the use of this service not only by the Nova component, but also by Glance or Swift. These components can be adapted to make use of the `SemanticAuthzService`. Thus, providing it as a service enables a unified authorization mechanism.

The access control system (`SemanticAuthzService`) has been developed in Java JDK 7, using the Java API for XML Web Services (JAX-WS) stack to provide a Web service interface. This component uses a DL reasoner (i.e. a reasoner using Description Logics). There are several suitable implementations, such as Pellet [24], Jena [25], and the one proposed by FaCt++ [26], among others. Pellet has been chosen as reasoner due to the fact that it supports high expressiveness dealing with OWL 2 ontologies and it is also able to perform incremental consistency checking. All the rules described in the section 4 are loaded in the reasoner as part of the bootstrapping process. Jena is also used to perform ontology operations and to manage the knowledge base since it is nowadays the de facto standard Java library to manage ontologies. The *Trust Manager* component described in our architecture uses Jena to enforce the trust relationships in the KB. Concretely, Jena version 2.6.2 and Pellet version 2.0.0 have been used. The trust relationships are specified using an external API that enables to define such relationships. They are stored in a trust database managed by the *Trust Manager* component itself.

For the proof of concept implementation, an interceptor has been placed in the EC2 IaaS API interface of the OpenStack Nova component. This IaaS API is the entry point to the IaaS services and receives all the requests to be served by the IaaS. Each request is checked against the `Authorizer` class of the EC2 *nova-api* module in order to determine if the user is or is not authorized to invoke the requested action in the IaaS. The `Authorizer` class has been modified to intercept the authorization requests and to forward them to the `SemanticAuthzService` which provides the authorization decision, accordingly. The modified python code of OpenStack modules has been adapted in order to execute the Java Web-service client library provided to connect to the

`SemanticAuthzService`. We have used JPype [3] for this purpose. JPype allows python programs to fully access java class libraries. Thus, we instantiate the Java Web-Service client library into Python and then we execute the appropriate methods from the Python code, accordingly. It should be noticed that we have adopted this approach for testing the client library purposes, as well as for simplicity during the development to achieve the performance metrics. However, a native Python client could be developed since the `SemanticAuthzService` interface is provided as a Web service, which is platform independent and highly compatible.

The monitoring module conceptually keeps the KB up-to-date with the current state of the IaaS. To do so, this module is continuously monitoring the IaaS to discover the status of the system and then updates the information model into the KB. In this proof of concept implementation, we have developed this module by attaching it to the message bus of Open Stack, which is the communication middleware used to coordinate all the components of the Open Stack architecture. This attachment enables to receive all the messages exchanged by Open Stack and update the KB of the `SemanticAuthzService` accordingly.

8.1. Testbed description

A testbed scenario has been deployed to test the proposed architecture and obtain some performance metrics. An OpenStack-based IaaS system has been deployed with 8 compute nodes to run virtual machine instances. Each node runs in a Hewlett Packard xw8400 Workstation with an Intel Xeon CPU at 3.00 GHz and 6 GB of RAM. The cloud controller, a Glance image service and the authorization service have been placed in one of these nodes. The purpose of using this configuration is to obtain tests for the worst case, in which the cloud controller and the access control system have to deal with the overload produced by running instances in the same host. Having a dedicated host for these components will result in lower execution times.

Three different operations have been considered in the tests. Namely, *describe instances*, *run instances* and *terminate instances*. They have been chosen as they represent the main operations related to virtual machine instance management. Moreover, they perform tasks with a different level of complexity. Thus, the *describe instances* operation is a lightweight operation, while the *run* and *terminate instances* perform some costly tasks, being the former even more costly. This makes the test offering more significant and different results. The aforementioned operations have been executed by invoking the corresponding *euca2ools* command, which uses the EC2 API to interact with the cloud controller.

In order to obtain statistics significant results, operations have been performed in sets of 100 executions, whose average is used as result value. The tests simulate the following use case. Firstly, a set of *describe instances* operations are executed, while the system is still not running any virtual machine instance, i.e. without instance overload. This enables to obtain the overhead produced by our authorization service in a dedicated system for the worst case, i.e. for the simplest operation with no instances. Then, the system is driven to a status in which each node is executing 4 instances (virtual machines). The rest of operation sets are then executed. These operations are executed alternating the set of *run instances* with another set of *describe instances* and the set of *terminate instances*. Operations are alternated in such a way that the system keeps in average the

[3] JPype is available at http://jpype.sourceforge.net/

overload of 4 instances per node. These executions provide results considering a constant system overload for the considered operations.

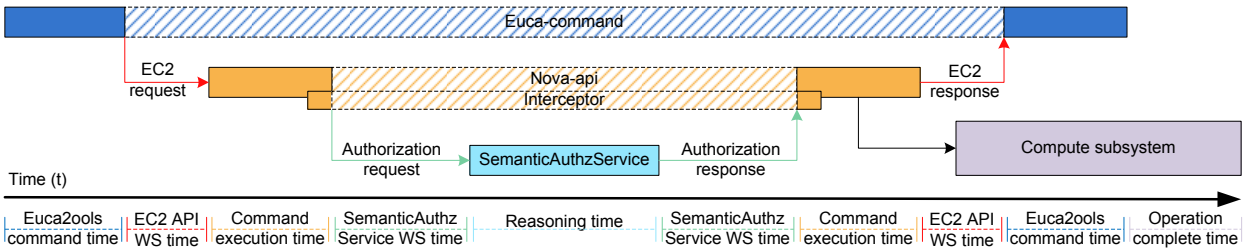


Figure 6: Operation execution workflow

Figure 6 shows the workflow followed by every operation and the average times taken for the entire executions. Firstly, the corresponding *euca2ools* command is executed. It accesses the EC2 API of the *nova-api* module, which is offered as a Web service. The authorization method is invoked in the *nova-api* and the interceptor redirects the request to the `SemanticAuthzService`. It performs the authorization decision and returns an access response to the interceptor. The interceptor enforces the authorization decision by allowing or forbidding the request in the *nova-api* module. Finally, the rest of the command is executed and a response is provided to the *euca2ools* command. At this point, some operations may still be pending due to its asynchronous nature. This is the case of the *run instances* and *terminate instances*. For these operations, the corresponding command finalizes just after launching the request to the compute system, but before the actual operation has been completed. For instance, the *run instances* command will trigger the operation and return before the instance is actually in the running state. For this reason, another time has been included in the test, which considers the time when the operation goal is achieved (e.g. the virtual machine is running).

The bottom side of the figure shows the times taken into consideration for analysis purposes. It should be noticed that the two times related to the same command execution or Web service invocation have been added and considered as a single time in the results. This is the case of the *euca2ools* and *nova-api* execution times, as well as the EC2 and Authorization request and response times. It should be also mentioned that only positive authorizations are considered for the tests. Otherwise, negative authorizations would not execute the command. This would distort the statistics significance of the results due to the short time of the executions.

The above scenario takes into account the overload of the IaaS system. But, there is also an internal overload the SemanticAuthzService should deal with. This is related with the number of elements the system should take into account to make authorization decisions, i.e. number of roles, users, permissions, objects, etc. These elements are contained in the KB used by the SemanticAuthzService. The elements contained in the KB are called individuals and the set of individuals for a specific execution is referred to as population. In order to study the scalability of the proposal, the above-described scenario has been executed 10 times using incremental populations. This allows getting results with an increasing order of authorization elements. A simulated monitoring component is in charge of producing different populations which represent different system statuses.

In order to quickly increment the number of individuals in the populations, this increment follows an exponential distribution. Finally, each individual is represented in the KB by a set of statements (not to be confused by authorization statements). Table 3 shows the number of individuals and its corresponding statements for each population considered during the tests. Note that the real size of the KB is determined by these statements and not by the number of individuals.

Population	1	2	3	4	5	6	7	8	9	10
Individuals	500	900	1,400	2,300	3,700	6,100	10,000	16,400	27,100	44,600
Statements	1,889	3,166	5,035	8,845	16,187	33,404	72,384	166,495	407,442	1,031,942

Table 3: Number of individuals and statements by population

8.2. Obtained results

Figure 7 shows the times obtained during the tests for each population. It shows times for the execution of the first *describe instances* operation with no instances running and the times for the three operations with 4 instances running in each node. Six different times are depicted in the figure. They correspond with the times taken from the workflow exposed in Figure 6.

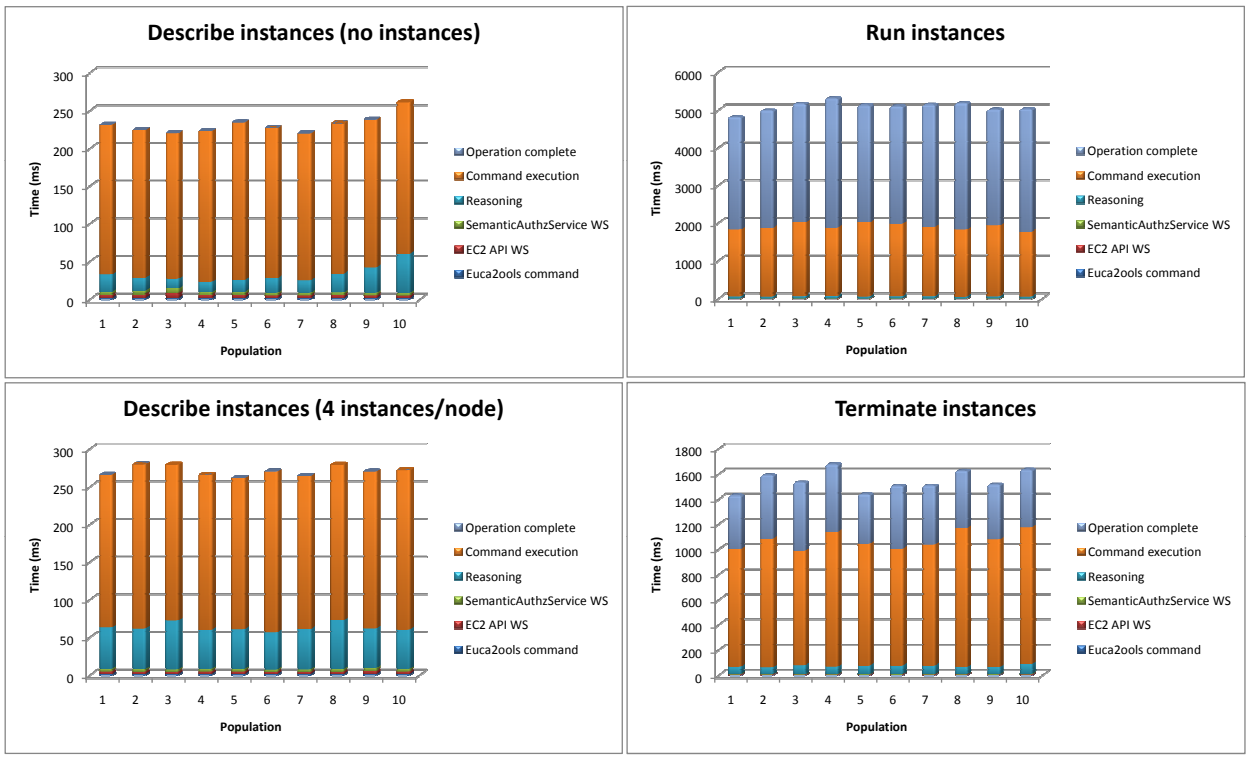


Figure 7: Operation execution times

As can be seen, the most part of the time is taken by the *Command execution* and *Operation complete* times. The *Reasoning* time spent by the authorization service is also outstanding in the

describe instances operations. But it should be taken into account that these operations are quite simple and the whole operation times are small. It should be noticed the different scales of the four figures. *Describe instances* operations take around 250 milliseconds, while *run instances* are about 5,000 and *terminate instances* around 1,500. The fluctuations observed in the different executions (and populations) are due to the fluctuating nature of the Cloud environment. Moreover, note that populations are following an exponential trend whereas the reasoning times are almost following a linear trend.

The figure also shows the difference between the *Command execution* and *Operation complete* times. While the *run instances* command finishes near the 1,900 ms (the time it takes to process the request), the underlying system takes around 3,000 ms more to actually get the virtual machine running. At this point, it should be noticed that these times have been taken considering a running system, i.e. discarding the first time an image is taken from the Glance image server, which takes much longer. For the rest of executions, the image is already cached in the node's hard disk and a copy-on-write image is quickly deployed. On the other hand, it can be also seen that the *describe instances* operation has no *Operation complete* time. That is because it is a get operation to retrieve information. Thus, the operation ends at the same time that the command terminates (synchronous execution), when it shows the requested information to the user.

In any case, it can be observed that the command and operation times are higher than the time taken by the access control system to perform an authorization decision. Trying to measure this difference, Figure 8 shows the overhead introduced by the access control system when compared with command and operation times. That is, the portion of the whole command or operation time that is due to the authorization process. Figure 8 on the left shows the overhead against the command time, while the right side shows it against the whole operation. The authorization time considered to calculate this overhead contains the *SemanticAuthzService WS* and *Reasoning* times.

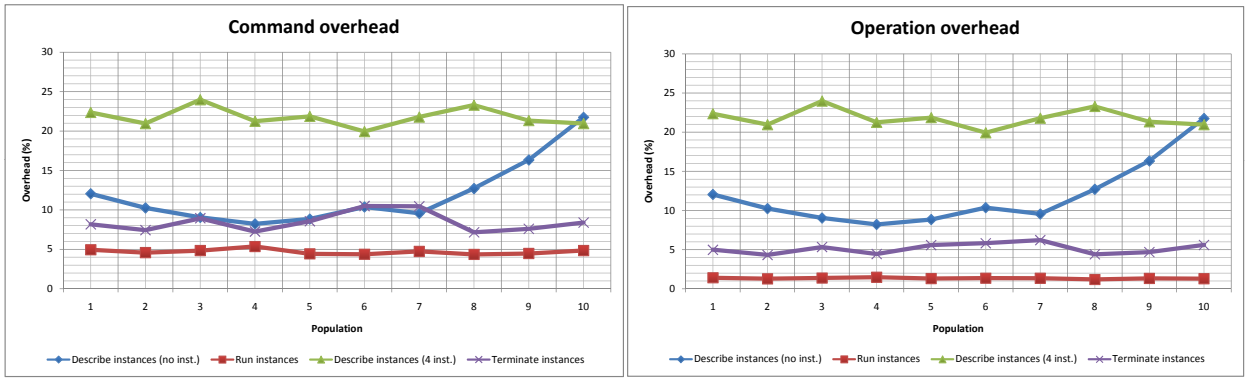


Figure 8: Overhead times

As can be observed, the overhead highly depends on the executed operation. This is due to the different ending times for each operation, as noticed in the previous figures. The average overhead is just around 2% and 5% for operations that manage the underlying infrastructure (5% and 9% if considering just the command execution).

The *describe instances* command shows the worst case for our authorization mechanism. For

such a simple command - which just performs a single quick database query - the overhead is still around 22%. The reason of this overhead is that this command takes very little time and it makes the authorization time to be more significant. However, note that this percentage is with respect to the total execution time which, in turn, is significantly lower for the *describe instances* than for others commands as the reader can see in figure 8. This fact causes that bigger overhead percentages can be tolerated in the system.

It can be observed that in an empty system with no instances running, the overhead trend is linear whereas it is almost constant for the rest of the cases. This is due to the external overload suffered by the host machine, which directly affects to the authorization service execution. Note that the execution of rest of the command is done while the VMs are being executed and this causes an important CPU overhead in the system which makes the overhead of the authorization system almost neglected, causing the constant trend shown. However, the *describe instances* command with no instances running do not suffer of this overhead and it causes a linear trend, according the increasing of the population.

The low execution time of the *describe instances* command with no instances running, enables to appreciate how increasing the population size affects the reasoning time. It can be seen that the last populations with a higher number of statements, also produce a higher overload. The exponential form of the graph is due to the population increments, which also follow an exponential distribution. Note that the worst case (22%) is for systems in which hundreds of thousands of VMs are being managed with a single instance of the access control system (more than 1 million of statements). For this kind of scenarios we'd probably have to adopt a replication of the access control system in order to balance the load. Anyway, for this big scenario, the authorization time is still provided in terms of milliseconds, which is a clear sign of the scalability of the proposed system.

It should be noticed that these overheads are provided without taking into account the time of the current Nova authorization mechanism. The overhead produced by our access control system when compared with a pure OpenStack deployment would be smaller, since just the increase over the Nova authorization time should be considered.

Finally, a constant tendency can be observed in the majority of the results shown in these figures. This tendency shows a good scalability of the proposed access control system, even if the population is increased following an exponential distribution.

9. Conclusions and Future Work

An authorization model suitable for cloud computing has been presented. This authorization model overcomes some of the limitations in expressiveness available in its predecessors and providing support for advanced authorization features such as RBAC, hRBAC, cRBAC, HO and authorization policies. Semantic Web technologies have been demonstrated as useful for describing authorization models. Moreover, the usage of the same language for expressing both information and authorization models avoid any mismatch between the semantics of both models. This, in turn, is a potential problem which appears in most of the current authorization proposals for Cloud computing. Advanced federation capabilities among tenants have been successfully supported. A fine-grained trust model for managing federations in Cloud computing environments has been also

provided. The complete architecture has been validated by means of a prototype implementation for the OpenStack platform using both Java and Phyton technologies. A performance analysis has provided prominent results.

As a future work, we expect to extend the integration with OpenStack in order to be more complete, providing interceptors not only for Nova but also for the Glance and Swift components of OpenStack. The idea is to achieve a complete authorization framework for OpenStack. An intensive performance analysis of the proposed authorization model is also expected in order to establish an analytical comparison of the trade-off between language expressiveness and system performance. Moreover, another expected step is the analysis and comparison of further advanced trust models and their suitability for Cloud computing.

References

[1] I. Foster, Y. Zhao, I. Raicu, S. Lu, Cloud Computing and Grid Computing 360-Degree Compared, in: Grid Computing Environments Workshop, IEEE, IEEE, 1–10, 2008.

[2] J. M. M. Perez, J. B. Bernabe, J. M. Alcaraz-Calero, F. J. G. Clemente, G. M. Perez, A. F. G. Skarmeta, Semantic-aware Authorization Architecture for Grid Security, Future Generation Computer Systems 27 (2011) 40–55.

[3] J. M. Alcaraz-Calero, G. M. Perez, A. F. G. Skarmeta, Towards an Authorization Model for Distributed Systems based on the Semantic Web, IET Information Security 4 (4) (2010) 411–421.

[4] J. B. Bernabe, J. M. M. Perez, J. M. A. Calero, F. J. G. Clemente, G. M. Perez, A. F. G. Skarmeta, Towards an Authorization system for cloud infrastructure providers, in: Internacional Conference on Security and Cryptography, 333–338, 2006.

[5] X.-Y. LI, Y. SHI, W. M. YU-GUO, MULTI-TENANCY BASED ACCESS CONTROL IN CLOUD, in: IEEE (Ed.), 2010 International Conference on Computational Intelligence and Software Engineering, vol. 1, IEEE, 1–4, 2010.

[6] D. Li, C. Liu, Q. Wei, Z. Liu, B. Liu, RBAC-based Access Control for SaaS Systems, in: IEEE (Ed.), 2010 2nd International Conference on Information Engineering and Computer Science, IEEE, 1–4, 2010.

[7] A. Sirisha, G. G. Kumari, API Access Control in Cloud Using the Role Based Access Control Model, in: IEEE (Ed.), Trendz in Information Sciences & Computing, IEEE, IEEE, 135–137, 2010.

[8] W.-T. Tsai, Q. Shao, Role-Based Access-Control Using Reference Ontology in Clouds, in: IEEE (Ed.), Tenth International Symposium on Autonomous Decentralized, IEEE, IEEE, 121–128, 2011.

[9] A. L. Pereira, RBAC for High Performance Computing Systems Integration in Grid Computing and Cloud C omputing, in: IEEE (Ed.), IEEE International Parallel & Distributed Processing Symposium, IEEE, IEEE, 914–921, 2011.

[10] J. Xu, T. Jinglei, H. Dongjian, Z. Linsen, C. Lin, N. Fang, Research and Implementation on Access Control of Management-type SaaS, in: IEEE (Ed.), The 2nd IEEE International Conference on Information Management and Engineering, IEEE, IEEE, 388–392, 2010.

[11] C. Danwei, H. Xiuli, R. Xunyi, Access Control of Cloud Service Based on UCON, LNCS Cloud Computing 5931 (2009) 559–564.

[12] J. Park, R. Sandhu, The UCON ABC usage control model, ACM Transactions on Information and System Security 7 (2004) 128–174.

[13] D. Fall, G. Blanc, T. Okuda, Y. Kadobayashi, S. Yamaguchi, Toward Quantified Risk-Adaptive Access Control for Multi-tenant Cloud Computing, in: The 6th Joint Workshop on Information Security, 1–14, URL `https://sites.google.com/site/jwis2011/program`, 2011.

[14] J. M. Alcaraz-Calero, N. Edwards, J. Kirschnick, L. Wilcock, M. Wray, Towards a Multi-tenancy Authorization System for Cloud Services, IEEE Security and Privacy 8 (6) (2010) 48–55.

[15] W. O. W. Group, OWL 2 Web Ontology Language: Document Overview, W3C Recommendation, W3C, 2009.

[16] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. T. andB. Grosof, M. Dean, SWRL: A Semantic Web Rule

Language Combining OWL and RuleML, Tech. Rep., W3C, URL http://www.w3.org/Submission/SWRL/, 2004.
[17] W. Bumpus, J. W. Sweitzer, P. Thompson, A. Westerinen, R. C. Williams, Common information model: implementing the object model for enterprisemanagement, John Wiley & Sons, Inc, 2000.
[18] T. Vetterli, A. Vaduva, M. Staudt, Metadata Standards for Data Warehousing: Open Information Model vs. Common Warehouse Metadata, ACM SIGMOD Record 29 (3) (2000) 68 – 75.
[19] M. Debusmann, A. Keller, SLA-driven management of distributed systems using the common information model, in: Proceeding of the 8th IFIP/IEEE International Symposium on Integrated Network Management, 1–14, 2003.
[20] H. Mao, L. Huang, M. Li, Web Resource Monitoring Based on Common Information Model, in: IEEE Asia-Pacific Conference on Services Computing, 520–525, 2006.
[21] D. Beckett, RDF/XML Syntax Specification, Tech. Rep., W3C, 2004.
[22] J. M. A. Calero, J. M. M. Perez, J. B. Bernabe, F. J. G. Clemente, G. M. Perez, A. F. G. Skarmeta, Detection of semantic conflicts in ontology and rule-based information systems", Data & Knowledge Engineering 69 (11) (2010) 1117 – 1137, ISSN 0169-023X, doi:10.1016/j.datak.2010.07.004, special issue on contribution of ontologies in designing advanced information systems.
[23] OpenStack, Open Source Cloud Computing Software, http://openstack.org, 2011.
[24] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, Y. Katz, Pellet: A practical OWL-DL reasoner, Journal of Web Semantics 5 (2) (2007) 51 – 53.
[25] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, K. Wilkinson, Jena: Implementing the Semantic Web Recommendations, in: Proceedings of the 13th international World Wide Web conference, ACM Press, 74–83, 2004.
[26] D. Tsarkov, I. Horrocks, Automated Reasoning, vol. 4130 of *Springer Lecture Notes in Computer Science*, chap. FaCT++ Description Logic Reasoner: System Description, Springer Berlin / Heidelberg, 292–297, 2006.