Usage Control on Cloud Systems

Enrico Carniani, Davide D'Arenzo,
Aliaksandr Lazouski, Fabio Martinelli, Paolo Mori

Istituto di Informatica e Telematica,
Consiglio Nazionale delle Ricerche
via Moruzzi, 1 - 56124 Pisa, Italy
email: {name.surname@iit.cnr.it}

Abstract

Cloud Computing is becoming increasingly popular because of its peculiarities, such as the availability on demand of (a large amount of) resources, even for a long time. For this reason, Cloud Computing represents a good solution for those companies that want to outsource part of their software processes. However, Cloud Computing introduces new security and management challenges with respect to traditional systems exposed on the Internet. This paper presents an advanced authorization service based on the Usage Control model to regulate the usage of Cloud resources, focussing on IaaS services.

Our framework addresses the issue of long lasting usage of resources, because it allows to define Usage Control policies which are continuously enforced while the access is in progress. In particular, our framework is able to interrupt the usage of such resources when the corresponding policy is not satisfied any more. In this paper, we present the architecture of the proposed framework describing the integration of a Usage Control based authorization service within one of the most popular software for running Cloud services: OpenNebula. Moreover, we describe the implementation of a prototype of the whole framework, along with some performance figures.

1 Introduction

The increasing popularity of Cloud systems is due to the on demand availability of big computational power for the execution of heavy parts of business and research processes [43, 37, 17]. As a matter of fact, Cloud providers allow their users to exploit a proper set of resources for their computation only when they actually need them. Cloud users, in turn, pay a fee depending on the resources they requested. Distinct Cloud service models have been defined by NIST [29], depending on the kind of resources that are provided. This paper is focused on the Infrastructure as a Service (IaaS) model, where the resources provided to users are computational infrastructures consisting of Virtual Machines (VMs) connected by virtual networks. When requesting VMs, users can choose the most proper network configuration, VM features (e.g., virtual CPU type and number, RAM memory and storage space), VM images (i.e., the operating system they need for their application). Moreover, users can install and run on the VMs allocated to them the applications they need. Once requested, the virtual computational infrastructure is available in a short time and the number of machines and/or their features can be updated by users (increased or decreased) on demand during the computation according to their needs. The other two Cloud service models defined by NIST are: Platform as a Service (PaaS), which provides an API for developing new services and a platform where these services can be deployed and executed, and Software as a Service (SaaS), where the provided resources are applications running on an existing Cloud infrastructure. The rest of the paper is focused on the IaaS model, although the approach proposed could be applicable to other Cloud service models. Usually, the instances of Cloud resources exploited by users are long-standing, and they are exposed to users through a proper interface. The VMs provided by an IaaS Cloud services are our reference example of long standing virtualized resources. In general, the Cloud user exploits them through a remote access using their IP addresses. IaaS Cloud facilities are currently provided by many big companies (Public Clouds), such as Amazon[1], Google[2], IBM[3], and Microsoft[4]. Alternatively, a number of Cloud frameworks, such as OpenNebula[5], Eucalyptus[6] or OpenStack[7], are currently available to deploy Cloud systems in users' data centres, to use their physical machines to host virtual ones (Private Clouds).

Besides the benefits previously described, the adoption of Cloud Computing to perform critical part of the business and research processes introduces also some problems, security being one of them. These security issues are described by the "European Network and Information Security Agency" (ENISA) in the report: "Cloud Computing. Benefits, Risks and Recommendations for Information Security" [32]. The "Cloud Security Alliance" (CSA) published two reports [1, 6] as well: "The Notorius Nine. Cloud Computing Top Threats in 2013" and "Security Guidance for Critical Areas of Focus in Cloud Computing" which, again, are focused on identifying the security problems peculiar of Cloud Computing. Some other research papers describe the main security issues in Cloud Computing such as [20, 51]. These documents point out that, besides the well-known threats of systems exposed on the Internet, the Cloud introduces further security challenges due to its specific peculiarities. These peculiarities include virtualization, multi-tenancy environment and long lasting accesses.

In this paper we propose an enhanced authorization service for Cloud IaaS services, which is able to continuously enforce security policies in order to interrupt accesses that are in progress when the corresponding access rights do not hold any more. This paper is an extension of our previous work, presented in [25], and our approach is based on the Usage Control (UCON) model, defined by Sandhu and Park in [35, 50]. In recent years, UCON has drawn a significant interest from the research community on formalization and enforcement of policies. There were several attempts to implement Usage Control, while the realization based on existing security standards is still an open issue. The design of an efficient and flexible framework (i.e., a policy schema, an architecture and an implementation) for Usage Control based on the OASIS XACML [33] standard is a challenge we address in our work.

1.1 Motivation and Contribution

The Usage Control model can be successfully adopted in the Cloud environment to regulate the usage of Cloud IaaS services because the accesses to those services could last for a long time, such as hours, days, or even more. Hence, some of the factors that have been evaluated by the security support to grant the initial access to the service could change while the access is in progress. As a consequence, it is possible that the right to access the service does not hold any more. For example, a computer science researcher R could request to a IaaS service provider the creation of a VM to host the Subversion server of his new three-years project. This VM will be used by all the project participants to manage the development of the project code, and it will be dismissed at the end of the project. Once created and deployed, the VM will be accessed by R using directly the public IP address (i.e., without any intervention of the Cloud service provider). R will configure the VM to allow the project participants to access the subversion service using directly the public IP address as well. Hence, after the creation request, no further request is sent to the Cloud service provider to use the VM. Let us suppose that the IaaS service provider allows users to run VMs only if their reputation is excellent. Adopting the traditional access control models, the value of the reputation of the user is controlled at request time only. Once the access has been granted and the VM has been started, this machine keeps on running until the user terminates it, even when the value of the user's reputation is not excellent any more. As a matter of fact, the VM creation and deployment requests are the only interactions among the Cloud service provider and the user, and no further controls on the reputation of the user are initiated by the security support during the VM life time. To address this issue, the Usage Control model enables the policy to state that some predicates that evaluate some mutable decision factors must be satisfied for the whole access time. This means that the access must be interrupted (or suspended) when these factors change in a way such that the policy is not satisfied any more. In the previous example, a predicate of the Usage Control policy could state that the user reputation must be excellent for all the VM life time. As soon as the reputation of the user decreases, his VM is suspended. Hence, the Usage Control approach prevents Cloud users from continuing

[1]http://aws.amazon.com/
[2]https://cloud.google.com/
[3]http://www.ibm.com/cloud-computing/us/en/
[4]https://azure.microsoft.com/en-us/
[5]http://opennebula.org/
[6]http://eucalyptus.com/
[7]http://openstack.org/

the use of resources that have been previously assigned to them as soon as the rights of using these have resources expired. This enhances the Cloud service security, avoiding that accesses are carried on when they become potentially dangerous. With reference to the previous example, if the reputation of the user R has decreased, it means that R has tried to perform a number of unauthorized operations, and most probably he will try to perform further malicious operations.

The main contributions of this paper are the following:

- The design of a complete framework for regulating the usage of Cloud IaaS services based on the Usage Control model. The paper provides a detailed description of the architecture of the proposed Usage Control service, focusing on the aspects concerning the implementation of the Usage Control model peculiarities, such as the continuous policy enforcement and the revocation of ongoing accesses;
- The integration of the Usage Control service within one of the most used tools for the provision of Cloud services, i.e., OpenNebula;
- A working implementation of the proposed framework;
- A set of experiments to evaluate the performance of our prototype.

Some attempt to adopt the UCON model in the Cloud have been proposed in the past (see Section 6). However, this paper represents a step forward because, to the best of our knowledge, none of the previous works presented the design and implementation of the overall authorization system architecture, and the integration within an existing Cloud framework.

1.2 Paper Structure

The paper is structured as follows. Section 2 gives a brief overview of the security support provided by two widespread Cloud frameworks, OpenNebula and OpenStack, and of the Usage Control model. Section 3 proposes our approach to regulate the usage of Cloud resources, presenting the detailed architecture of the proposed framework, describing the integration with OpenNebula, and also giving some examples of Usage Control policies in our reference scenario. Section 4 presents a working implementation of the whole framework, and Section 5 shows some performance figures from our experiments. Section 6 describes several related works. Finally, Section 7 briefly discusses the applicability of our framework to other scenarios, while Section 8 concludes the paper.

2 Background

This section briefly reports some background notions on the authorization systems supported by two widespread Cloud management tools, OpenNebula and OpenStack, and on the Usage Control model, on which the proposed framework relies.

2.1 Access Control in the Cloud

Security is a critical issue in Cloud because, besides the usual issues of a system exposed on the Internet, additional ones arise due to virtualization and multi tenancy [19, 22, 38, 39]. Access control is crucial, in order to avoid unauthorized access to systems and to protect organizations assets [46]. The adoption of several kinds of access control mechanisms have been proposed in the literature to protect Cloud resources (see Section 6), starting from simple Access Control Lists to complex authorization systems based on security policy languages. OpenNebula and OpenStack manage their authentication and authorization phases as follows:

- OpenNebula[8] (ONE) is a fully open source Cloud management tool which offers a simple but feature-rich and flexible solution to build and manage Clouds and virtualized data centres [30, 42]. The authentication phase could exploit different well known techniques, such as user-password,

[8] http://opennebula.org/

LDAP [47], as well as Sunstone[9] and Server Authentication as EC2[10]. The OpenNebula native authorization system supports access control list (ACL) and usage quotas. ACL regulates the access to Cloud resources based on the user ID and role, while the usage quotas regulate how many resources the user can consume

- OpenStack[11] is a Cloud computing framework, released under Apache 2.0 license, which allows the deployment and management of public, private and hybrid Cloud environments. Authentication is provided by the Keystone[12] component, which is the OpenStack Identity Service. Keystone supports several different authentication methods: LDAP [24], HTTPD[13], X.509 [15] and other self-built systems. In particular, Keystone also supports federated access[14]. For authorization, Keystone only supports one mechanism based on RBAC like policies. However, it is possible to extend the basic Keystone authorization support as shown in [40]. On the other hand, Swift[15], which is Object Storage service, supports access control lists. Finally, Congress[16] is a recent OpenStack project aimed at providing a Cloud service which enforces policies on collections of Cloud services. Congress allows Cloud providers to declare, monitor, enforce, and audit policies in their heterogeneous Cloud environment. The language exploited for expressing Congress policies is Datalog. Such policies are declarative, i.e., they define which states of the Cloud are allowed, where the state of the Cloud is represented by the data gathered from the collection of Cloud services monitored by Congress. The Congress architecture is modular, and it allows to monitor any Cloud service by simply defining a proper driver to collect information from it.

2.2 The Usage Control Model

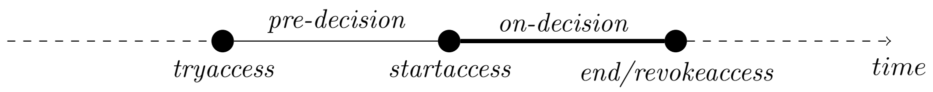


Figure 1: **Access and Usage Control**

The Usage Control model (UCON), defined in [35, 50], encompasses and extends the existing access control models. In particular, UCON introduces new features in the decision process w.r.t. traditional Access Control models, such as the *mutability of attributes* and the *continuity of policy enforcement.* These features are meant to guarantee that the right of a subject to use a resource holds not only at access request time, but also while the access is in progress. In the following, we recall the UCON core components.

Subjects and Objects The subjects are the entities who exercise their rights on the objects by performing actions on them. In our scenario, the subjects are the Cloud users who exploit the VMs (objects) provided by Cloud IaaS services to perform their business or research tasks.

Actions The actions represent the operations performed by the users on the objects. In our scenario, Cloud users perform actions on VMs such as create, deploy and resume.

Attributes Attributes are paired to subjects, objects, actions and environment to describe their features. An attribute is *immutable* when its value can be updated only through an administrative action. An example of immutable attribute is the subject's role in RBAC based systems, which is updated by the system administrator, for instance, as a consequence of a career advancement. Instead, an attribute is *mutable* when its value changes over time because of the normal operation of the system. Some mutable

[9]http://docs.opennebula.org/4.12/administration/sunstone_gui/sunstone.html#sunstone
[10]http://docs.opennebula.org/4.12/advanced_administration/public_cloud/ec2qcg.html#ec2qcg
[11]https://www.openstack.org/
[12]http://docs.openstack.org/developer/keystone/
[13]http://httpd.apache.org/
[14]http://docs.openstack.org/security-guide/identity/federated-keystone.html
[15]http://docs.openstack.org/developer/swift/
[16]https://congress.readthedocs.org/en/latest/

attributes change their values as a consequence of the policy enforcement, because the policy includes attribute update statements that can be executed before (*pre-update*), during (*on-update*), or after (*post-update*) the execution of the action. As an example, let us consider the mutable attribute which represents the number of running VMs deployed by a subject on a Cloud IaaS service. The value of this attribute changes over time because it is incremented by a *pre-update* statement in the policy when the subject is authorized to perform the action of deploying a new VM, and it is decremented by a *post-update* statement in the policy when the subject terminates an existing VM. Another example of mutable attribute concerning the actions is the number of instances of the same action that are currently in execution. Mutable attributes can change their values also because of other actions performed by the subjects which are not regulated by the usage control policy. For instance, the attribute that describes the physical location of the subject changes when he moves from one place to another. Some mutable attributes change their values for both the reasons described before. For example, the balance of the subject's e-wallet could decrease because the Usage Control policy includes a *pre-update* statement which states that the subject must pay for deploying a new VM, while it could increase when the subject deposits some money in his e-wallet through a bank transaction. Finally, other attributes change their values independently of the user behaviour. For instance, date, time and CPU load, are attributes of the environment which belongs to this last set.

Authorizations Authorizations are predicates that evaluate subject and object attributes and the requested right to decide whether the subject is allowed to access the object. The evaluation of the authorization predicates can be performed before executing the access (*pre-authorizations*), or continuously while the access is in progress (*on- authorizations*) in order to promptly react to mutable attribute changes.

Conditions Conditions are environmental or system-oriented decision factors, i.e., dynamic factors that do not depend on subjects or objects. Hence, the evaluation of conditions involves attributes of the environment and of the action, and it can be executed before (*pre-conditions*) or during (*on-conditions*) the execution of the action.

Obligations Obligations are decision factors which verify whether a subject has satisfied some mandatory requirements before performing an action (*pre-obligations*), or whether a subject continuously satisfies these requirements while performing the access (*on-obligations*). Obligations can be enforced after the execution of the action as well (*post-obligations*) but, in this case, they cannot affect the execution of the action.

Continuity of Policy Enforcement The attribute mutability introduces the necessity to execute the Usage Control policy evaluation process continuously while an access is in progress. This is because the values of the attributes that previously authorized the access could change in a way such that the access right does not hold any longer. In this case, the access is revoked as soon as the policy violation is detected.

The Usage Control model can be successfully adopted in case of long-standing accesses because the decision process consists of two phases (Figure 1). The *pre-decision* phase corresponds to traditional access control, where the decision process is performed at the request time to produce the access decision. The *on-decision* phase, instead, is executed after the access is started and implements the continuity of control, which is a specific feature of the UCON model. Continuous control implies that policies are re-evaluated each time mutable attributes change their values. The *pre-decision* process evaluates *pre-authorizations*, *pre-conditions*, and *pre-obligations*, and if a policy violation is detected the access is not permitted. The *on-decision* process continuously evaluates *on-authorizations*, *on-conditions*, and *on-obligations*. In this case, when a policy violation is detected the ongoing access is revoked and the resource is released. For further details about the Usage Control model please refer to [26, 31, 36].

2.3 U-XACML: an Extension of XACML for Usage Control

In order to express usage control policies, we defined an extension of the XACML language, called U-XACML. XACML is a standard developed by the OASIS consortium for expressing and managing access control policies in a distributed environment. Briefly, the XACML standard defines a policy metamodel, syntax, semantics, and the related enforcement architecture. The top-level element of a policy is <*PolicySet*>, which includes a set of <*Policy*> elements, each of which, in turn, includes a <*Target*>, which denotes the target of the policy, and a set of <*Rule*> elements which represent the authorization rules. A rule is defined by three main components: the <*Target*>, the <*Condition*>, and the effect of the rule which can be either Permit or Deny. The <*Target*> denotes the target of the rule, i.e., to which authorization requests the rule can be applied. The <*Condition*> elements are predicates evaluating the attributes. Optionally, a rule can include also the <*ObligationExpression*> element. A rule is applicable to an access request if the target of the access request matches the target of the rule and if all the conditions included in the rule are satisfied. If a rule is applicable to an access request, the effect declared for the rule determines whether the access request is permitted or denied, and the related obligation must be executed. For a detailed description of the XACML standard please refer to [33].

XACML allows to express traditional access control policies and it does not have specific constructs to express the continuity of policy enforcement. Hence, the U-XACML language extends XACML with some constructs for usage control as follows. To represent the peculiarity of the Usage Control model, i.e., the continuity of policy enforcement, the U-XACML language allows to specify in the <*Condition*> element a clause, called *DecisionTime*, which defines when the evaluation of this condition must be executed. The admitted values are *pre* and *on* denoting, respectively, *pre-decisions* and *on-decisions*. In this way, the conditions whose decision time is set to *pre* are usual XACML conditions. Instead, the conditions whose decision time is set to *on* must be continuously evaluated while the access is in progress. We recall that XACML conditions are exploited to represent both UCON authorizations and conditions. In the same way, U-XACML extends the <*ObligationExpression*> element with the *DecisionTime* clause to define when the obligation must be executed. In this case too, the admitted values for the *DecisionTime* clause are: *pre* (*pre-obligations*, i.e., usual XACML obligations) and *on* (*on-obligations*), and *post* (*post-obligations*).

To deal with mutable attributes, U-XACML introduces a new element, <*AttrUpdates*>, which represents the attribute updates in the policy. This element includes a number of <*AttrUpdate*> elements to specify each update action. Each <*AttrUpdate*> element also specifies when the update must be performed through the clause *UpdateTime* which can have one of the following values: *pre* (*pre-update*), *on* (*on-update*), and *post* (*post-update*). An alternative approach, proposed in [23], considers the attribute updates as a special type of obligations which are performed by the system. For instance, the authors of [49] use XACML to specify usage control policies, and they exploit XACML obligations to express attribute updates in the policy. However, at the level of the U-XACML policy, we prefer to keep the attribute updates separated from the obligations which actually represent constraints which must be fulfilled before or during the action in order to authorize the execution of the action itself. For further details on the U-XACML language, we refer to [10].

3 Usage Control on Cloud

This paper proposes the adoption of an advanced authorization service based on the Usage Control model (called Usage Control service) to regulate the usage of resources in Cloud IaaS services. The Usage Control service enforces the Usage Control policy defined by the Cloud provider exploiting the U-XACML language (see Section 2.3). The main feature of the proposed system is that the Usage Control policy, taking into account factors that change over time, is continuously enforced while these resources are in use. As a consequence, the access of a given user to the IaaS service is interrupted as soon as the related policy is not satisfied any more. Obviously, the resource usage revocation should be implemented in such a way that the results produced up to that time are not destroyed, but they can be retrieved subsequently by the user.

The users of the Cloud IaaS service exploit the providers' resources by creating and starting new Virtual Machines, resuming Virtual Machines which have been suspended, creating Virtual Networks, connecting Virtual Machines to Virtual Networks, and so on. The action listed before will be referred in the following as *security relevant actions*, because they are relevant from the point of view of the security of the Cloud IaaS service. In fact, they allow users to exploit the hardware resources of the Cloud provider (CPU time, RAM memory, permanent storage, network bandwidth, etc.), and to access and use software resources of the Cloud service (VM templates, data, etc.). Moreover, an important feature of some of these actions is that they are long lasting, which means that they allow Cloud users to exploit the IaaS service for some time, from a few minutes to hours, days, or even years. As an example, a researcher could create and start a set of VMs that host the management and the development environments at the beginning of a research project, and run these VMs until the project has been completed (e.g., after 3 years). In fact, since the time required to users to set up VMs with their working environments (i.e., create and start a VM, install the tools and load the data required for executing their tasks) is not negligible, users typically exploit IaaS Clouds for long lasting computations. Hence, the execution of these actions represents a security threat for the Cloud service provider, because the factors that initially allowed the execution could change while the action is still in progress. Besides wasting Cloud resources, a malicious or unauthorized user could perform a Denial of Service (DoS) attack by creating and running a very large number of VMs (thus filling, for example, the available RAM memory). Moreover, a malicious user could exploit his VMs to perform attacks to other VMs on the same Cloud or to external machines.

Summarizing, the Usage Control service can be successfully integrated in the Cloud environment to enhance the security of IaaS services, because the accesses to those services represent a relevant source of threats for the Cloud provider since they could last for a long time, such as hours, days, or even more. Finally, the proposed system would contribute to address some of the threats described in [1]. In fact, the continuous enforcement of the Usage Control policy and the interruption of running VMs when users violates the policy could avoid the prosecution of attacks in case of account hijacking and/or abuse of Cloud Services.

3.1 Examples of Usage Control Policies

This section presents three simple examples of Usage Control policies for the scenario of interest, focussing on operations on VMs. These policies could be part of a more complex set of policies adopted by a Cloud IaaS service to regulate the usage of VMs. Since the U-XACML language is too verbose, for the sake of readability, in this paper we use a human-readable language to show Usage Control policies. However, the translation from this human-readable language to U-XACML, which is the language actually enforced by the Usage Control service, is straightforward. Moreover, since the framework proposed in this paper is part of the Coco Cloud project[17], the policy authoring tool developed within this project can be exploited to easily create Usage Control policies. A description of this policy authoring tool can be found in [8].

In our scenario, we suppose to have three roles for subjects: guest, customer, and administrator. Moreover, we suppose that a subject cannot hold more than one role at the same time. A guest is a subject who registered to the Cloud service, and who can exploit the Cloud resources for free for an evaluation period, with some constraints. A customer, instead, is a user who pays a monthly fee for exploiting the Cloud IaaS service, and no limitations are imposed on the resources he can use. An administrator is an employee of the Cloud provider who manages the Cloud IaaS service.

The policies are shown in Listings 1, 2, and 3, where s represents the subject requesting the operation to the Cloud service, o denotes the resource that s wants to access, e.g., a VM, and a is the action that s wants to execute on o. The attributes exploited by these policies are described in Table 1.

Listing 1 First example of Usage Control policy

```
$policyA :$
    $target :$
        $(o.type = "VM")$
    $pre - authorization :$
        $(a.id = "deploy")$    $AND$
        $("guest" \in s.role)$    $AND$
        $(s.id = o.owner)$    $AND$
        $(o.requiredMemory \leq 4GB)$    $AND$
        $(s.numVMs = 0)$    $AND$
        $(s.reputation = "excellent")$
    $pre - update :$
        $(s.numVMs++)$
    $on - authorization :$
        $(s.reputation = "excellent")$
    $post - update :$
        $(s.numVMs--)$
```

[17]http://www.coco-cloud.eu/

Attribute name	**Description**	**Mutable**
s.id	Unique ID of the subject *s*	No
s.role	Role of the subject *s*	No
s.reputation	Reputation of the subject *s*	Yes. It decreases, for instance, as a consequence of a policy violation
s.numVMs	Number of VMs deployed by the subject *s* on the Cloud IaaS service	Yes. It changes every time the subject *s* creates a new VM or terminates an existing one
s.unpaidFees	Number of overdue fees, i.e., the number of monthly fees that the subject *s* should have paid but he has not paid yet	Yes. It changes when the subject *s* omits to pay a monthly fee, or when *s* pays one (or more) overdue fees
s.clearance	Security level granted to an administrator *s* of the Cloud IaaS service	Yes. It changes, for instance, when the administrator is disciplined for some reason
o.type	Type of the object *o*, e.g., *VM*	No
o.owner	ID of the subject owner of the object *o*	No
o.requiredMemory	Amount of memory required for executing the object *o* of type *VM*	No
a.id	Unique ID of the action *a*	No

Table 1: **Attributes exploited in the examples of Usage Control policies**

The goal of *policyA*, shown in Listing 1, is to regulate the deployment of VMs by guest users. It imposes limitations on the memory size and on the number of concurrent VMs that a guest user can execute, and it ensures that only guest users with excellent reputation are allowed to deploy and run VMs. Hence, *policyA* checks that the reputation of a guest user is excellent both at request time, in order to authorize the deployment of the VM, and continuously during the execution of this VM, in order to suspend it as soon as the reputation is not excellent any more. In particular, this policy is applicable to operations performed on Virtual Machines, because the *target* section of the policy states that the resource type must be equal to *VM* (line 3). The *pre-authorization* section checks that the action ID is equal to *deploy* (line 5), and that the user holds the role *guest* (line 6). Moreover, the *pre-authorization* section controls that the user who requests to deploy the VM is the owner of that VM by matching the attribute which represents the ID of the subject with the attribute *o.owner* of the object (line 7), and that the amount of RAM memory to be assigned to the VM to be deployed (*o.requiredMemory*) is equal or less than 4GB (line 8). Since the user ID, the user role, the VM owner and the VM memory size are immutable attributes, they are checked only once in the *pre-authorization* phase, and they will not change their values during the new VM lifetime. In fact, we assume that a VM must be stopped before changing its memory size. The *pre-authorization* section also includes a predicate concerning a mutable attribute of the user, *s.numVMs*, which represents the number of running VMs deployed by the user (line 9). The value of this attribute is incremented in the *pre-update* section (line 12), i.e., when the VM deployment has been authorized, and it is decremented in the *post-update* section (line 16), i.e., when the VM has been terminated. Although it is a mutable attribute, its value is checked only in the *pre-authorization* section because, when a guest user is already running one VM, this policy prevents him from creating another VM. Moreover, in order to allow only guest users whose reputation is excellent to deploy and carry on the execution of VMs, *policyA* checks that value of the attribute *s.reputation* is equal to "excellent" both in the *pre-authorization* and in the *on-authorization* sections (lines 10 and 14, respectively).

Listing 2 Second example of Usage Control policy

```
1:  $policyB :$
2:      $target :$
3:          $(o.type = "VM")$
4:      $pre-authorization :$
5:          $(a.id = "deploy")$   $AND$
6:          $("customer" \in s.role)$   $AND$
7:          $(s.id = o.owner)$   $AND$
8:          $(s.unpaidFees = 0)$
9:      $on-authorization :$
10:         $(s.unpaidFees \leq 1)$
```

The second policy, *policyB*, is shown in Listing 2, and it concerns the deployment of VMs by regular customers. The purpose of *policyB* is to permit the execution of VMs only to customers who are in good standing with the payment of the monthly fee. The *target* section of this policy states that the resource type must be equal to *VM* (line 3), i.e., the policy is applicable to operations performed on Virtual Machines. The *pre-authorization* predicate checks that the action ID is equal to *deploy* (line 5), that the user who requests to start the VM holds the customer role (line 6), and that this user is the owner of that VM (line 7), similarly as the previous policy. The policy allows a customer to deploy a new VM only if he has paid all the overdue fees, and his running VM is suspended if he skips the payment of the monthly fee for more than 1 time. To this aim, the *pre-authorization* section includes a predicate which checks that the value of the attribute *s.unpaidFees* is zero, i.e., the user *s* must not have any overdue fee still to pay in order to be authorized to deploy a new VM (line 8). Moreover, the *on-authorization* section includes a predicate (line 10) which checks that the number of unpaid fees is less than or equal to 1. This means that the policy suspends the VMs running on behalf of the customer *s* when *s* omits to pay the monthly fee of the Cloud IaaS service for two times.

Listing 3 Third example of Usage Control policy

```
1:  $policyC:$
2:      $target:$
3:          $(o.type = "VM")$
4:      $pre-authorization:$
5:          $("administrator" \in s.role)$      $AND$
6:          $(s.clearance = MAX)$
7:      $on-authorization:$
8:          $(s.clearance = MAX)$
```

The last policy of our example, *policyC*, is shown in Listing 3. This policy allows the administrators of the Cloud IaaS service to perform any operation on any VM of the system. However, this permission is valid only if and as long as the clearance level of the administrator is maximum. In fact, the clearance of an administrator could be decreased when the administrator is disciplined for some reason. In this case, the administrator is not allowed to perform new operations on the VMs, and the operations he started that are still in progress are interrupted. Similarly to the previous policies, the *target* section checks that the resource type is equal to VM. The *pre-authorization* section includes two predicates. The first predicate (line 5) checks that the subject holds the administrator role. The second predicate (line 6) checks that the clearance level of the administrator is maximum in order to grant to the subject the right of initiating an action on a VM. Since no constraints are specified on the action ID, *policyC* is applicable to all operations on VMs. Moreover, the clearance level of the administrator is continuously controlled in the *on-authorization* section by the predicate in line 8, in order to ensure that the user holds the maximum clearance level also while the operation is performed.

3.2 Integration of the Enforcement Infrastructure within Cloud Providers

The architecture of the proposed framework for supporting the enforcement of Usage Control policies in Cloud IaaS services is shown in Figure 2, and it is derived from the XACML reference architecture [33]. The main components of the architecture are the *Cloud provider* (on the left) and the *Usage Control service* (on the right).

The architecture consists of three horizontal layers:

- The Application Layer (AL) holds the Cloud IaaS service and other services that manage some of the attributes needed for policy evaluation. In Figure 2, the components of this layer are represented by the blue boxes;
- The Usage Control Layer (UCL) holds the core components of the Usage Control system, which are represented by the green boxes in Figure 2. On the Cloud provider side, it includes the Policy Enforcement Point, while on the Usage Control service side it hosts all the components devoted to the (continuous) evaluation of Usage Control policies;
- The Communication Layer (CL) is the lower end of the infrastructure that performs the communication between the components on the Cloud provider and the Usage Control service. In Figure 2,

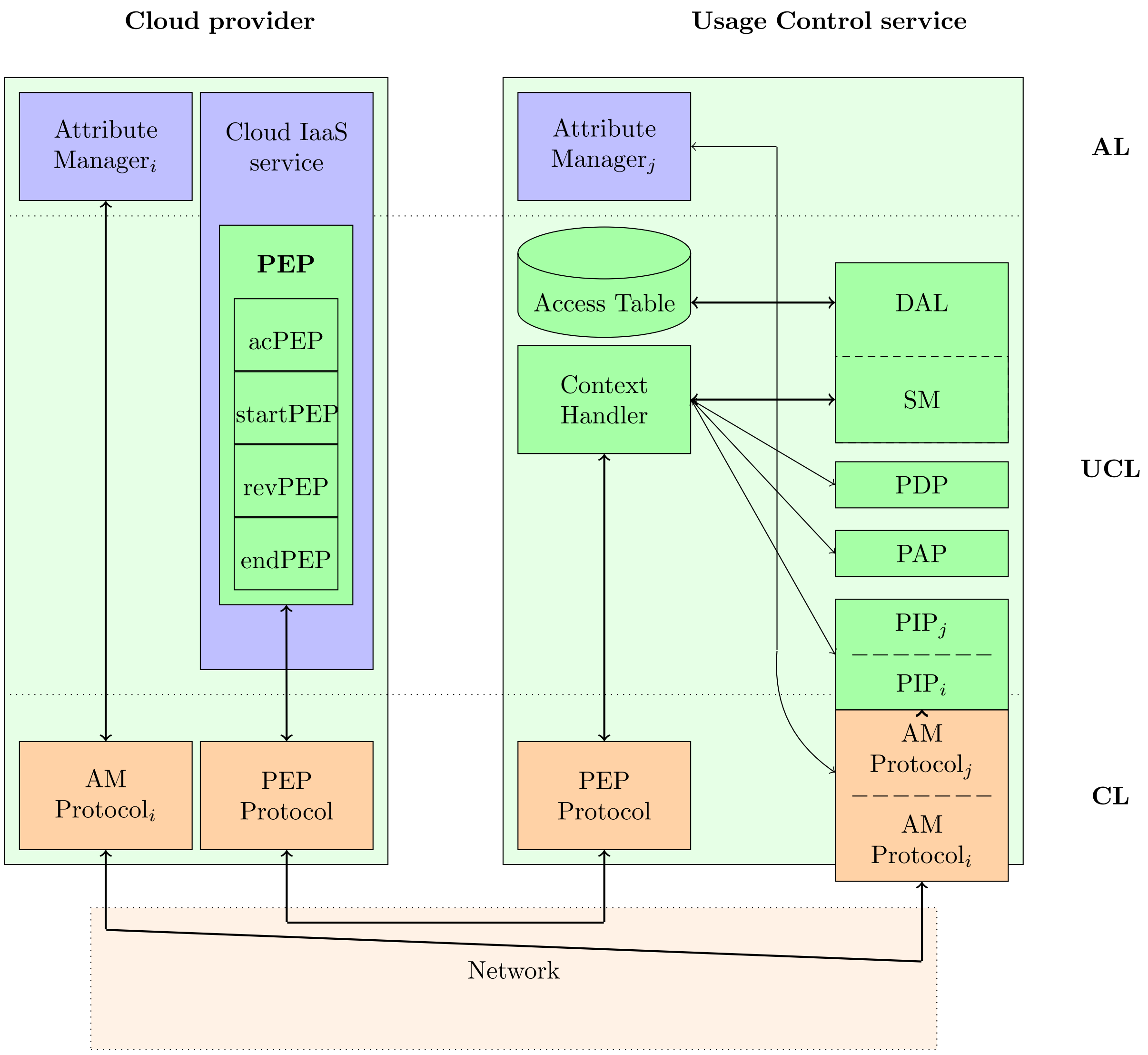


Figure 2: **Architecture for the enforcement of Usage Control policies**

the components of the Communication Layer are represented by the orange boxes.

The enforcement of Usage Control policies requires to embed a Policy Enforcement Point (PEP) within the Cloud IaaS service in order to intercept and regulate the execution of each security relevant action of the scenario. In particular, a PEP consists of four components: the acPEP, the startPEP, the endPEP and the revPEP. The acPEP is in charge of performing the *pre-decision* phase. The startPEP, the endPEP and the revPEP, instead, are in charge of managing the *on-decision* phase. In practice, each of these components is a distinct software module that must be properly embedded in the code of the Cloud IaaS service.

The protocol which regulates the interactions between the PEP components and the Usage Control service is composed of the following messages, derived from the *Usage Control actions* described in [50]:

- $tryaccess(s, o, a)$: sent by the acPEP to the Usage Control service when the subject s requests to execute the access (s, o, a), i.e., s requests to perform the security relevant action a on the virtual machine o;
- $permitaccess(s, o, a)$: sent by the Usage Control service to the acPEP as response to the $tryaccess(s, o, a)$ message to allow the access (s, o, a);
- $denyaccess(s, o, a)$: sent by the Usage Control service to the acPEP as response to the $tryaccess(s, o, a)$ message to deny the access (s, o, a);
- $startaccess(s, o, a)$: sent by the startPEP to the Usage Control service when the access (s, o, a) started;
- $endaccess(s, o, a)$: sent by the endPEP to the Usage Control service when the access (s, o, a) terminates;
- $revokeaccess(s, o, a)$: sent by the Usage Control service to the revPEP to suspend an ongoing access (s, o, a).

The workflow of the interactions between the PEP components and the Usage Control service is shown in the sequence diagram in Figure 3 (where areq represents the access request (s, o, a)). When the Cloud IaaS service tries to execute a security relevant action a, the acPEP intercepts it and suspends its execution. The acPEP retrieves the information related to this access (such as the ID of the subject s who is performing the action, the ID of the resource o on which the action is performed, etc.). Then, the acPEP builds the *tryaccess* message exploiting the data previously collected, and sends it to the Usage Control service (interaction 1:tryaccess(areq) in Figure 3). The Usage Control service performs the *pre-decision* process and returns the result to the acPEP which enforces it. If the Usage Control service decides to allow the action, it sends the *permitaccess* message to the acPEP (interaction 2:permitaccess(areq)), which resumes the execution of a. Instead, if the Usage Control service sends the *denyaccess* message, the acPEP skips the execution of the action. Let us suppose that the execution of a is permitted. In this case, when the execution of a has started, the startPEP sends the *startaccess* message to the Usage Control service to initiate the *on-decision* phase (interaction 3:startaccess(areq)). Hence, while a is in progress, the Usage Control service continuously checks that the Usage Control policy is satisfied. In case this policy is violated, the Usage Control service sends the *revokeaccess* message to the revPEP (interaction 4a:revokeaccess(areq)), asking for the termination of a. The revPEP is properly configured for gracefully interrupting each security relevant action without compromising the involved data or computations. For instance, if the security relevant action is the execution of a VM, the revPEP simply suspends this VM when it receives the *revokeaccess* message. In this way, the Cloud provider could temporary restart this VM to allow the owner to recover his data. Instead, if a is terminated by a user, the endPEP sends the *endaccess* message to the Usage Control service (interaction 4b:endaccess(areq)) to stop the *on-decision* phase for that action. Please notice that our protocol includes a further action, *startaccess*, w.r.t. the *Usage Control actions* described in [50]. This action is meant to explicitly start the *on-decision* process.

3.2.1 Integration within OpenNebula

In this paper we focus on Cloud IaaS services based on OpenNebula, because it is well suited for integration with other software components and extends to almost all the hardware available on the market [42, 30].

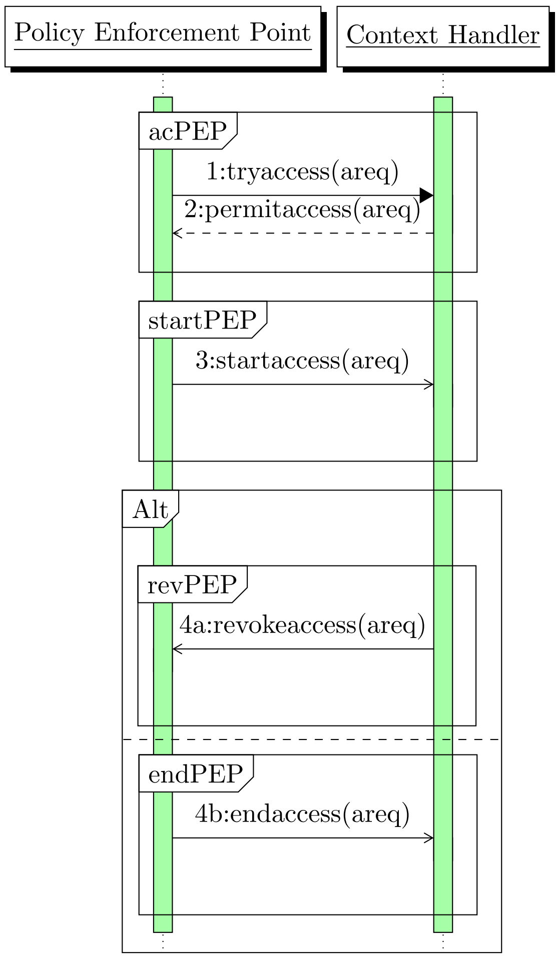


Figure 3: **Interactions between Policy Enforcement Point and Context Handler**

onevm command	**Authz Operation**	**VM State**	**UCON Action**
deploy	MANAGE/Request	$\rightarrow$ PENDING	*tryaccess*
resume	MANAGE/Request	STOPPED $\rightarrow$ PENDING	*tryaccess*
-	-	PENDING $\rightarrow$ RUNNING	*startaccess*
shutdown, delete	MANAGE/Request	RUNNING $\rightarrow$ SHUT-DOWN	*endaccess*
stop, suspend	MANAGE/Request	RUNNING $\rightarrow$ STOPPED	*endaccess*

Table 2: **Mapping of OpenNebula commands to *Usage Control actions***

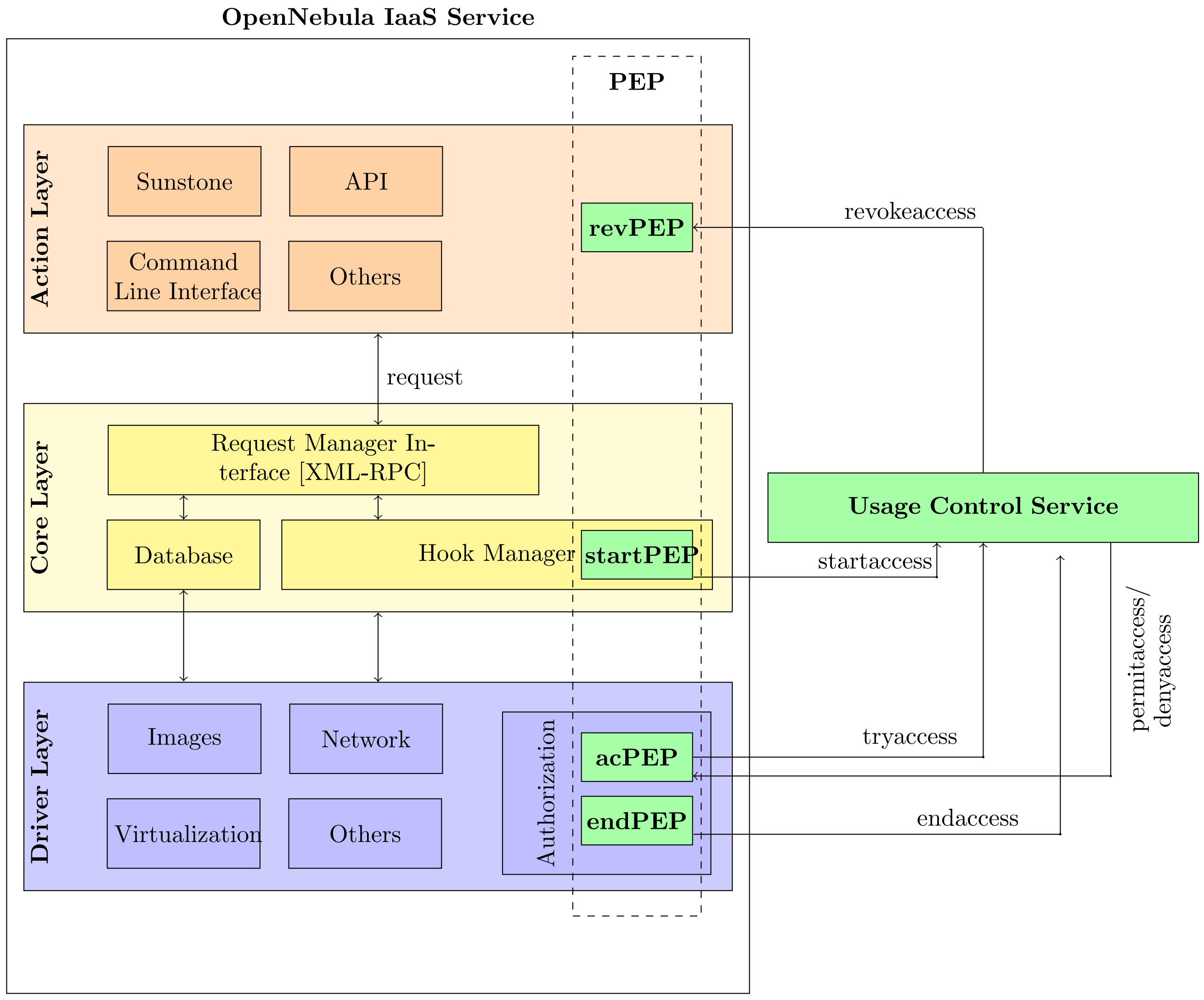


Figure 4: **Integration of a Policy Enforcement Point in the OpenNebula architecture**

Figure 4 shows how the Usage Control framework, represented by the green boxes, has been integrated with the OpenNebula architecture[18]. In particular, the acPEP and the endPEP have been embedded in the Authorization Driver (AD), the startPEP has been integrated in the Hook Manager (HM) and the revPEP has been deployed in the Action Layer.

The Authorization Driver is a component of OpenNebula that is meant to be configured to adopt external authorization systems (such as the Usage Control service). The Hook Manager is component which can be configured to execute a program (called hook) when a change in the state of a VM or a Host happens. Choosing the execution of a VM as example of security relevant action, Table 2 shows the mapping between the OpenNebula commands invoked by the Cloud user to manage a VM and the *Usage Control actions* that are sent by the PEP components to the Usage Control service. The first column specifies the command sent by the user to the OpenNebula gateway in order to manage a VM. The OpenNebula Core intercepts this command and sends the proper access request, represented in the second column of the table, to the Authorization Driver (AD) for the security check. The third column defines the state of the VM before and after the execution of the command. The last column specifies the corresponding *Usage Control action*.

When a user invokes the *onevm deploy* command through the OpenNebula interface (e.g., using the Command Line Interface, CLI), the OpenNebula Core intercepts it and sends the *MANAGE* request to

[18]http://docs.opennebula.org/4.12/_images/overview_integrators.png

the AD, the state of the requested VM is set to *PENDING*. In turn, the acPEP embedded in the AD sends the *tryaccess* message to the Usage Control service. If the acPEP receives back the *denyaccess* message from the Usage Control service, the *onevm deploy* operation is not executed. Instead, if the *permitaccess* message is received, the AD grants the access. In this case, the OpenNebula Virtualization Driver executes the deploy command, and the state of the VM changes from *PENDING* to *RUNNING*. Consequently, the startPEP embedded in the HM is executed and it sends the *startaccess* message to the Usage Control service. When the user wants to stop his running VM, he sends to OpenNebula one of the following commands: *onevm shutdown, delete, stop, suspend*. When the corresponding *MANAGE* request is sent to the AD, the acPEP sends the *endaccess* message to the Usage Control service. Instead, if the security policy is violated during the VM execution, the revPEP embedded in the Action Layer of OpenNebula receives the *revokeaccess* message from the Usage Control service to suspend the VM execution. In this case, the revPEP performs the invocation of the *onevm suspend* command acting as *admin* user and this changes the state of the VM from *RUNNING* to *STOPPED*. In this way, the data produced on the VM by the user's applications are preserved, and the Cloud provider could enable its user to recover them simply by temporary restarting the suspended VM. Notice that the AD has been configured to allow the suspend command executed by the *admin* user.

3.3 Usage Control Service

This section presents the architecture of the Usage Control service, which is the advanced authorization service we designed to support the enforcement of Usage Control policies. This architecture is shown on the right side of Figure 2. As previously stated, it is derived from the XACML reference architecture, which has been extended to include the components required for the evaluation of U-XACML Usage Control policies and the management and revocation of the ongoing usage sessions. The architecture components are the following.

3.3.1 Attribute Managers (AMs)

Attribute Managers are the components which manage the attributes of subjects, resources, environment and actions, allowing to retrieve and to update their values. They could run in the Usage Control service, on the Cloud provider side, or they could even be external, i.e., they could run in other administrative domains. Some existing services can be exploited as Attribute Managers. For instance, OpenNebula can be considered as an Attribute Manager since it can provide information about both Cloud users and resources.

3.3.2 Policy Information Points (PIPs)

Policy Information Points are interfaces for interacting with Attribute Managers in order to perform the following operations on attributes: *retrieve, subscribe/unsubscribe,* and *update*. In general, the attributes required for the evaluation of a Usage Control policy could be managed by several Attribute Managers, which require different protocols for interacting with them, and which provide different functionalities. Hence, PIPs mimic a plugin architecture to let the Usage Control service be as flexible as possible in interacting with distinct and different Attribute Managers. In particular, the proposed architecture includes a set (chain) of PIPs. Each PIP provides the same interface to the Context Handler (*retrieve, subscribe/unsubscribe,* and *update*), while it implements the specific protocol to interact with a given Attribute Manager and the specific algorithm to perform the requested operation. For instance, if the Attribute Manager of attribute *attr* does not support subscription, the PIP paired with *attr* should implement the subscription mechanism in order to provide the *subscribe* interface. This PIP could invoke periodically the Attribute Manager to retrieve the updated value in order to compare it with the previously collected value. If the new value is different from the previous one, the PIP notifies the component which performed the subscription (i.e., the Context Handler). The time interval between two consecutive queries to the Attribute Manager is a configuration parameter and it is set according to the attribute to be monitored. More complex techniques could be implemented. For instance, the previous PIP could exploit risk based techniques to decide how much time it should wait before retrieving the next attribute value from the Attribute Manager. The *retrieve* interface could be implemented by simply forwarding the request to the Attribute Manager, or by exploiting more complex techniques in order to reduce the time required for the attribute retrieval phase. For instance, a PIP could return the value retrieved in a previous interaction instead of retrieving a fresh value from the Attribute Manager when it is aware that the attribute value is not changed (e.g., when the PIP itself requested to the Attribute Manager to lock the attribute).

3.3.3 Policy Decision Point (PDP)

The Policy Decision Point is a XACML evaluation engine that takes a policy and an access request as input, evaluates the policy for that request, and returns the decision. Inputs and outputs of the PDP are represented in the format defined by the XACML standard.

3.3.4 Session Manager (SM), Database Abstraction Layer (DAL), and Access Table (AT)

The Session Manager represents an extension with respect to the XACML reference architecture. It is in charge of keeping trace of the usage sessions, in order to implement the continuous enforcement of Usage Control policies while the accesses are in progress. The SM exploits the Access Table to store data about ongoing sessions. The AT can be seen as a DataBase where each entry refers to an ongoing session and contains the session ID, the access request, the session status (i.e., pending, active, revoked, ended), and other data. The SM allows to create a new entry in the AT to represent a new usage session through the *create* operation, to change the session status through the *updateStatus* operation, and to remove an entry related to a terminated access through the *remove* operation. Each entry is paired with a unique ID. Moreover, the SM provides a further operation, *query*, which allows to get the list of the active sessions involving a given subject, object, and/or action. Since the Usage Control service has been designed to be highly configurable, a relational Data Abstraction Layer [3] has been inserted in the architecture. This allows the choice of the DataBase implementation for the AT which best meets the needs of the specific scenario when deploying a new installation of the Usage Control service.

3.3.5 Context Handler (CH)

The CH is the front-end of the Usage Control service, and it coordinates the interactions with the components of this service in order to perform:

- the *pre-decision* phase, which produces the access decision;
- the *on-decision* phase, which continuously enforces the ongoing policy while the access is in progress;
- the *post-decision* phase, which executes the post updates of attributes.

The CH manages the protocol for communicating with the PEP components embedded in the Cloud IaaS service exploiting the *Usage Control actions* defined in [50] according to the workflow described in Section 3.2.

In the *pre-decision* phase, the CH receives from the acPEP the *tryaccess* message which includes the access request (*areq*), and it interacts with the other components of the service to evaluate the *pre-policy* (i.e., the policy including *pre-authorization, pre-conditions, pre-updates* and *pre-obligations* only) in order to produce the decision and send back the response. The *pre-decision* phase workflow is shown in the sequence diagram of Figure 5. In particular, the CH retrieves the *pre-policy* from the Policy Administration Point (PAP) which acts as policy repository (interaction 2:get(pre-policy) in Figure 5). Then, the CH interacts with the PIPs to retrieve all the attributes required for the decision process, i.e., the attributes related to the user who requested the access, to the Cloud resource, to the environment and to the action (interaction 3:retrieve(areq)). The CH exploits the *retrieve* interface provided by the PIPs sending the access request as parameter, and it receives back the same access request enriched with the retrieved attributes. The Usage Control service embeds a chain (i.e., an ordered set) of PIPs, and the CH invokes them in the predefined order. Once all the PIPs have been invoked, the CH sends the *pre-policy* along with the enriched access request to the PDP, which evaluates the policy for that request and returns the access decision (interaction 4:evaluate(policy,areq)). Let us suppose that the access is permitted. The access decision also includes a list of attribute updates, that the CH executes by exploiting the *update* interface of the PIPs (interaction 5:update(uplist)). These updates correspond to the *pre-update Usage Control actions*. Finally, the CH invokes the SM through the *create* operation to create a new entry for this access (interaction 6a:create(areq)); the session status of the new entry is "pending". The SM enriches the access request adding the session ID, and the CH returns the *permitaccess* message to the acPEP (interaction 7a:permitaccess(areq)).

The *on-decision* phase starts when the CH is notified by the startPEP (through a *startaccess* message) that the previously allowed access has began. The *on-decision* phase consists of a first evaluation of the *on-policy* (i.e., the policy including *on-authorization, on-conditions, on-updates* and *on-obligations*), followed by a number of re-evaluations of this policy every time the value of an attribute changes. The workflow of the first evaluation of the *on-policy* is very similar to the workflow of the *pre-policy* evaluation. First, the CH retrieves the *on-policy* from the PAP. Next, it retrieves the updated values of the attributes required for the decision process exploiting the *subscribe* interface of the PIPs. Then, it asks the PDP to evaluate the *on-policy* on the access request enriched with the collected attributes. Let us suppose that the PDP allows the access. Hence, the CH invokes the *update* interface of the PIPs to send them the list of attribute updates included in the response of the PDP. Each PIP executes the updates related to the attributes it manages according to the update statements it receives. Finally, the CH changes the session status to "active" interacting with the SM through the *updateStatus* command.

Due to the attribute subscriptions, the CH will be also notified by the PIPs when the value of one of the attributes involved in the ongoing decision process, say *attr*, has changed. In this case too, the CH coordinates the other components of the Usage Control service in order to perform the *on-policy* re-evaluation. The workflow is shown in Figure 6. When the CH is notified by the PIP, because *attr* changed its value (interaction 1:notifyUpdate(attr) in Figure 6), it invokes the SM through the *query* operation to get the list of active accesses whose execution right could change because of the new value of *attr* (interaction 2:query(attr)). Then, the CH retrieves the *on-policy* from the PAP (interaction 3:get(on-policy)), and re-evaluates it for each session of the list. To this aim, the CH retrieves the current values of the required attributes from the PIPs (interaction 4:retrieve(areq)), and it invokes the PDP for the policy evaluation (interaction 5:evaluate(policy,areq)). If the *on-policy* is satisfied, the access continues, otherwise it must be revoked. The response returned by the PDP includes a list of updates of attributes. In case the policy is satisfied, the updates returned by the PDP correspond to the *on-updates Usage Control actions*, while they correspond to the *post-updates Usage Control actions* in case of access revocation. The CH delegates these attribute updates to the PIPs exploiting the *update* interface (interaction 6:update(uplist)). To revoke the access, the CH sends the *revokeaccess* message to the revPEP (interaction 7:revokeaccess(areq)). In addition, the CH cancels the attribute subscriptions related to this access by invoking the *unsubscribe* operation of the PIPs (interaction 8:unsubscribe(areq)), and it changes the status of the entry in the AT related to this access to "revoked" by invoking the *updateStatus* operation of the SM (interaction 9:updateStatus(areq,revoked)).

If an access terminates because the user stopped it, the CH is notified by the endPEP through the *endaccess* message. In this case the CH changes the status of the related entry in the AT to "ended", and it executes the post updates of attributes, which correspond to the *post-updates Usage Control actions.*

3.4 Concurrent Management of Attributes

When multiple Usage Control services run concurrently and exploit a set of common attributes, some concurrency issues in attribute management could arise if two or more decision processes concurrently retrieve and update the same attribute. Such concurrency issues could arise, for instance, in a federated Cloud. A Cloud Federation [7, 11] is an infrastructure which allows a set of Cloud providers to offer their services through a unified platform. A major advantage of a Cloud Federation is that it provides elasticity beyond the scale of the single Cloud provider. For instance, it allows its users to exploit services from distinct Cloud providers at the same time.

In a federated Cloud, each provider runs its own Usage Control service which is configured to exploit both attributes managed by the provider itself, as well as shared attributes provided by the Cloud Federation. In fact, the Cloud Federation sets up its own Attribute Manager to deliver to Cloud providers a set of attributes which represent some features of subjects and objects at Federation level. The total number of VMs running on behalf of a user s on the providers of the Federation, $s.numVMs$, is an example of mutable attribute of the subject which is provided by the AM of the Federation. The AM which manages the attributes of the Federation are considered external for the Cloud providers. In addition, other existing services running in the Cloud Federation can be exploited by the Usage Control services of the Cloud providers as external AMs. For instance, the Cloud Information System of the Federation could provide attributes concerning each Cloud provider, while the Monitoring Systems and the Accounting DataBase

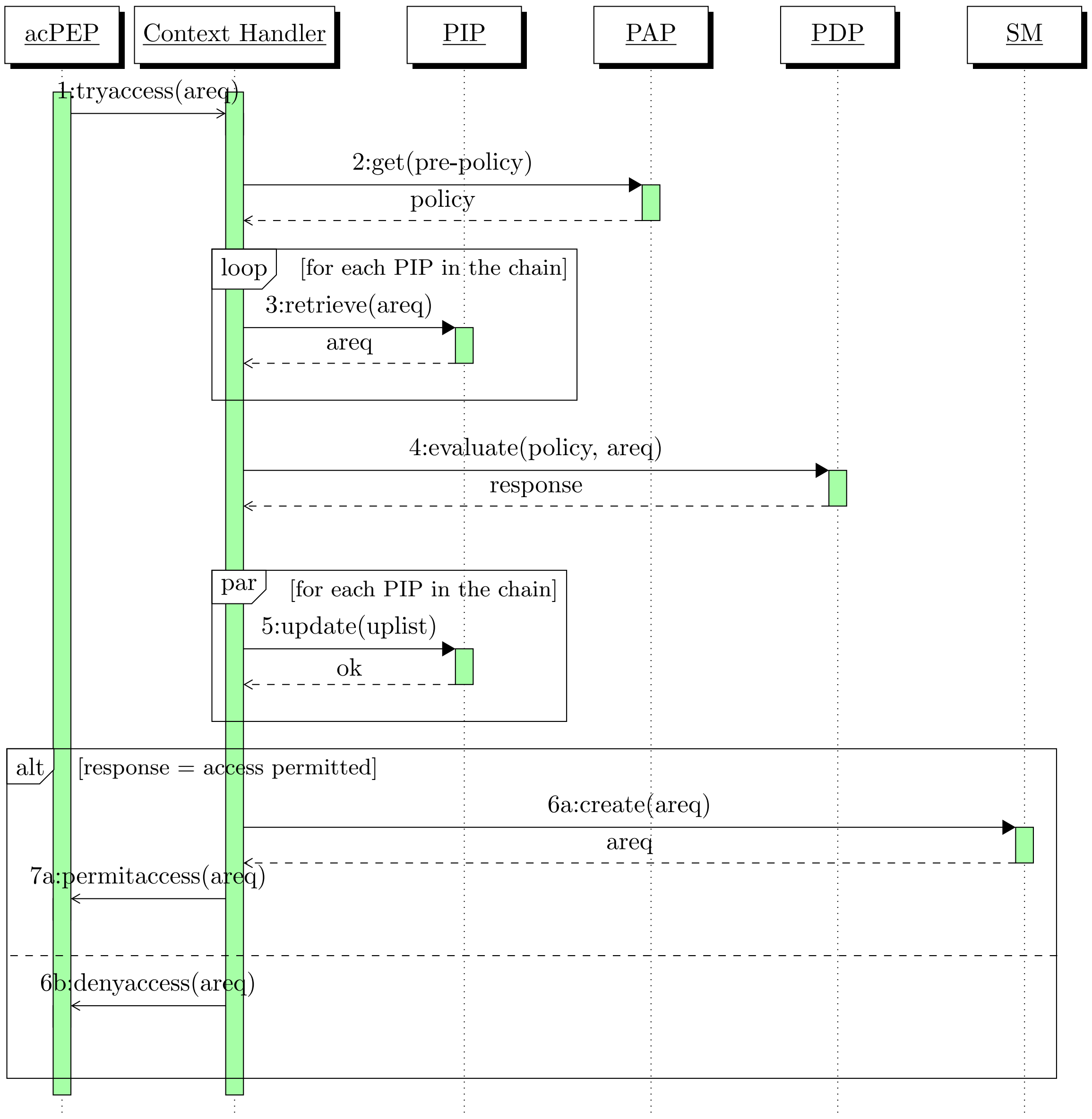


Figure 5: **Workflow of the *pre-decision* phase**

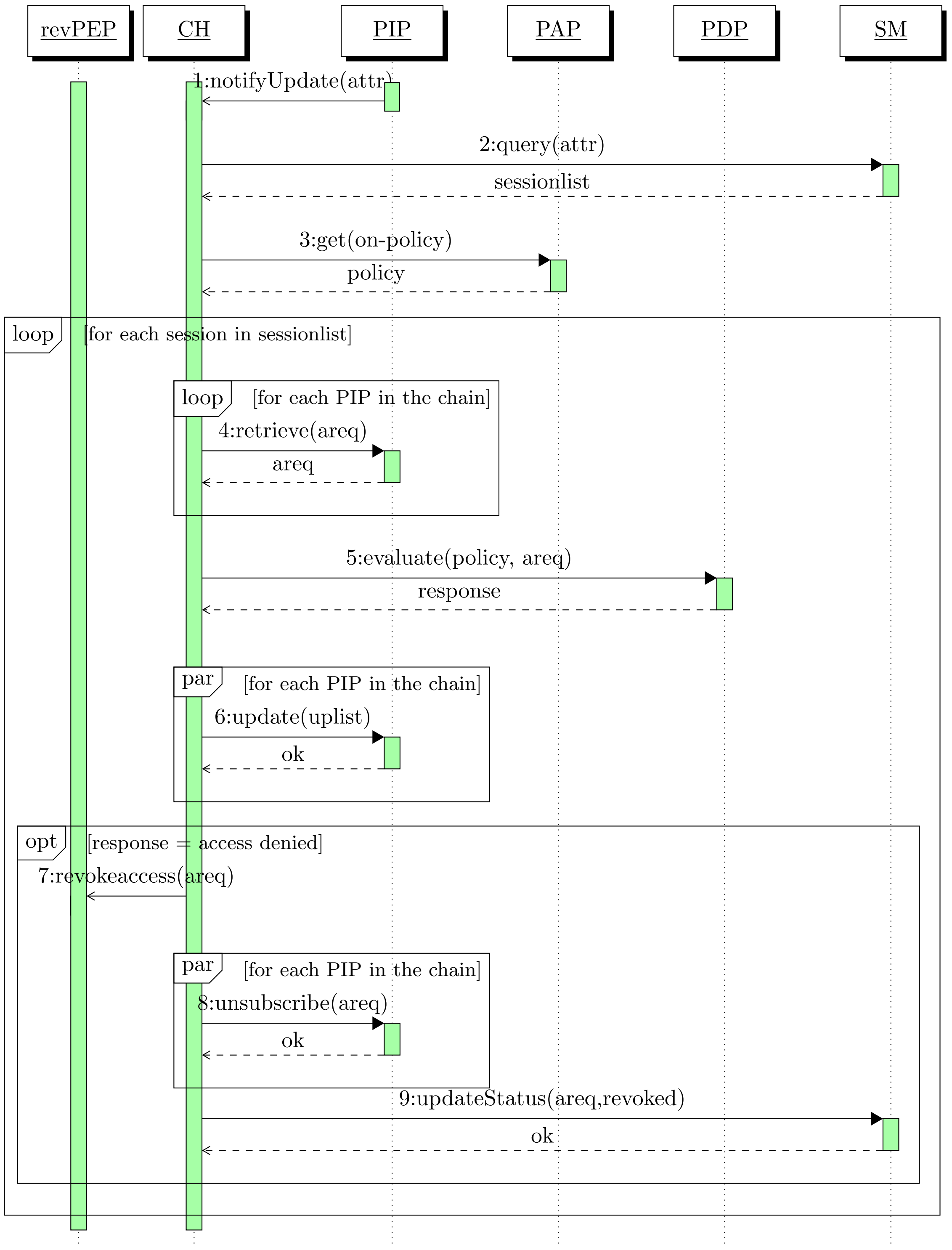


Figure 6: **Workflow of the *on-policy* re-evaluation**

of the Federation could provide attributes concerning the resource consumption of each subject.

Let us suppose that each Cloud provider of the Federation enforces a policy which allows a user s to deploy a new VM only if the number of VMs currently in execution in the Federation on behalf of s is less than N. If s tries to deploy two VMs on two distinct Cloud providers of the Federation at the same time, two decision processes are executed concurrently by the Usage Control services of the two providers. These decision processes retrieve the attribute *s.numVMs* from the AM of the Federation. If the current value of the attribute is less than N, they allow the deployment of the new VM, and they ask the AM of the Federation to store the increased value of *s.numVMs*. A *race condition* arises when the result of two or more decision processes depend on the interleaving of the *retrieve* and *update* operations they performed. *Lost update* and *inconsistent retrieval* are examples of the race condition problem. With reference to the previous example, a lost update arises when the two decision processes read the value of the *s.numVMs* simultaneously, i.e., they read the same value because each of the two processes misses the attribute update performed by the other. If the initial value of the attribute *s.numVMs* is N-1, both decision processes will allow the deployment of a new VM, while only one VM should be deployed according to the policy. Moreover, both the decision processes update the value of *s.numVMs* setting it to N. Hence, the resulting value of *s.numVMs* will be N, while it should be N+1 because two VMs have been deployed. The two-phase locking protocol (2PL) [4, 34] is a standard solution to address race conditions in attribute retrieval and updates. It pairs each attribute with a lock mechanism, which should be supported by the related AM. When a decision process requires to read and update one or more attributes, all these attributes are locked before performing the operations. If an attribute is already locked, the decision process is suspended until the lock is released. It means that another concurrent decision process involving the same attribute is in progress. As soon as the decision process has been performed, the involved attributes are unlocked. To avoid deadlocks, attributes must be locked in a predefined order by all the Usage Control services. Hence, we assume that the AMs are able to properly perform the lock and unlock operations, and that it is possible to define the attribute ordering because the set of attributes supported by the Usage Control services is known a priori. Obviously, the adoption of a locking mechanism introduces a delay in the attribute retrieval phase. In fact, when a decision process requires to access an attribute that is already locked, it is delayed until the attribute is available.

4 Prototype Implementation

This section describes the implementation details of the components of the Usage Control service and of the PEP enabling the integration within OpenNebula. All the components of the Usage Control service have been developed in Java, while the acPEP, the startPEP, the endPEP, and the revPEP have been developed exploiting the Ruby language. With respect to our previous work described in [25], we dramatically improved most of the code of our framework in order to improve its performance.

4.1 PIP Implementation

As explained in Section 3.3, PIPs can be seen as plugins which allow the Usage Control service to interact with a set of distinct and different AMs through the same interface. This interface is implemented by each PIP providing the remote methods for the *retrieve, subscribe/unsubscribe,* and *update* operations exploiting XML-RPC over HTTP. In our prototype, HTTPS is not required because the CH and the PIPs are deployed on the same machine. For the purpose of validating our framework, we implemented three PIPs:

- Session counter: this PIP returns the number of active sessions of a given user, exploiting the Session Manager as (local) Attribute Manager;
- Local DB PIP: this PIP manages attributes paired with users or VMs which are stored on the DataBase of the Usage Control service.
- Remote DB PIP: this kind of PIP allows to exploit remote DataBases as Attribute Managers. In particular, it implements SSL-based connections to MySQL servers in order to retrieve attributes paired with users or VMs.

4.2 PDP Implementation

The PDP engine has been implemented as Eclipse project using WSO2 Balana API[19]. Balana is an open source XACML implementation based on Sun-xacml. This engine supports several features, such as XACML 3.0, Maven[20] support for compile with unit tests, and decision profile support. Moreover, this engine has excellent performance.

4.3 DAL and SM Implementation

The DAL has been implemented exploiting the Oracle Java Persistence Architecture (JPA) v2.1 [5], which is an Object Relational Mapping (ORM) abstraction with its own query language (called JPQL). We have chosen the EclipseLink [9] implementation of the JPA-ORM instead of the usual Hibernate, since the former is more compliant with the JPA recommendations. The relational DataBase Management System (DBMS) which has been adopted in our framework is Apache Derby[21]. However, a JPA compliant system allows to easily substitute the DBMS by simply modifying a configuration file. Apache Derby is implemented in Java, and the performance of the whole system takes advantage of this choice. In fact, other popular relational database management systems[22], such as MySQL server or SQLite with their own JDBC, perform worse than Derby. The SM implementation is based on the *javax.persistence* and the *org.eclipse.persistence.annotations* Java libraries. Finally, the DAL and SM implementation is fault-tolerant. In fact, the data related to the ongoing sessions resist to system crashes, and they are resumed when the Usage Control service is restarted.

4.4 CH Implementation

The CH is the front-end of the Usage Control service and it interacts with the PEP components embedded in OpenNebula. To this aim, the CH exploits the remote procedure call mechanism provided by the Apache XML-RPC library[23] version 3 to expose the *tryaccess, startaccess*, and *endaccess* operations which build up the communication protocol of the Usage Control service. Since in our testbed the Usage Control service and OpenNebula are deployed on distinct machines, the XML-RPC protocol is over HTTP Secure (HTTPS) to protect the communication. However, our implementation completely hides these communication details by means of a Java abstract interface (represented by the box PEP Protocol in Figure 2). In this way the acPEP, the startPEP, and the endPEP embedded in OpenNebula simply invoke the methods of this interface (called *tryaccess, startaccess*, and *endaccess* as well) to communicate with the CH. Symmetrically, the revPEP implements the *revokeaccess* procedure and exposes it through the XML-RPC protocol over HTTPS to allow the CH to send the access revocation message.

4.5 Implementation of acPEP, startPEP, endPEP, and revPEP.

The code required to implement the PEP functionalities described in section 3.2 has been embedded in several components of OpenNebula. In particular, the Authorization Driver (AD) is responsible for sending the *tryaccess* and *endaccess* messages and enforcing the corresponding *permit/denyaccess* messages. The Hook Manager (HM), instead, is in charge of sending the *startaccess* message (see Figure 4).

When the user issues the *onevm deploy* command, the OpenNebula Core calls the AD for the access decision before the command execution. We configured OpenNebula to exploit our implementation of AD, which is written in Ruby and embeds the code of the acPEP and of the endPEP. The AD receives the user id (*user-id*), the VM id (*vm-id*), and the command to be executed (*type-req*). Next, the AD gets from the OpenNebula DataBase the state of the requested VM. If the state is *PENDING*, then the AD recognizes that the user tries to start the VM (see Table 2) and sends the *tryaccess* message to the Usage Control service. As previously described, this communication is implemented through the XML-RPC protocol over HTTPS. In particular, the acPEP invokes the *tryaccess* remote method exposed by the CH, passing the XACML Request as parameter of the method. The Usage Control service replies with the

[19]http://xacmlinfo.org/category/balana/
[20]http://maven.apache.org/
[21]http://db.apache.org/derby/
[22]http://www.jpab.org/All/Query/All.html
[23]https://ws.apache.org/xmlrpc/

access decision. The AD processes the response and replies with *SUCCESS* to the OpenNebula Core if the *permitaccess* is received. Otherwise, the AD triggers the authorization exception.

When the VM has been started, its state changes to *RUNNING* and this event triggers the HM. Again, we configured OpenNebula to exploit our version of HM (which is a ruby script as well), which embeds the code of the startPEP. Hence, the HM sends the *startaccess* message to the Usage Control service by invoking the corresponding remote method exposed by the CH through the XML-RPC protocol. When the user decides to stop his VM, he sends this command to the OpenNebula Core, which forwards it to our AD. Our AD executes the code of the endPEP which sends the *endaccess* message to the Usage Control service exploiting the same protocol used for the *tryaccess* one.

For what concerns the access revocation, the revPEP is implemented as a server waiting for revocation messages from the Usage Control service. In this case too, the XML-RPC protocol over HTTPS has been adopted, and the Usage Control service invokes a function (*def* in Ruby) called *revokeaccess* that is exposed by the revPEP. This function issues the *onevm suspend* command to the OpenNebula Core acting as *admin* user. This request is forwarded to the AD which authorizes it because it was originated by the *admin* user, and the VM is suspended by the OpenNebula core.

5 Performance Evaluation

To validate the proposed framework, we performed a set of experiments exploiting the previously described prototype to measure the performance of the Usage Control service. In particular, we estimated the time required by the Usage Control service to perform the initial access decision, and to decide whether some of the running VMs must be suspended according to the Usage Control policy as a consequence of an attribute update. In our testbed, the Usage Control service was deployed on a machine equipped with a CPU Intel(R) Xeon W3565 @3.20GHz and 8GB of RAM memory, while the OpenNebula framework was deployed on another machine equipped with a CPU Intel(R) i5-760 @2.80GHz and 8GB of RAM memory. These machines were connected by a Gigabit Ethernet local network.

The first set of experiments measured the delay introduced by our framework in the deployment of new VMs due to the execution of the *pre-decision* process, varying the number of attributes evaluated by the enforced policy. In other words, we measured the time required by the Usage Control service to process the *tryaccess* message. The attributes exploited in these experiments were provided by a local AM, i.e., a DataBase running on the same machine as the Usage Control service. Figure 7(a) shows the results of our experiments performed by varying the number of attributes from 1 to 20, while Figure 7(b) reports the results varying the number of attributes from 20 to 1,000. The pseudo-code exploited to perform these measurements is shown in Listing 4, and it was executed on the machine running the Usage Control service. Each value was measured 1,000 times, and we computed the average. The variance was negligible.

Listing 4 Pseudo-code for attribute based measurements

```
Require:
    $S : \{tryaccess\}$
    $CONDITION : \forall\ requests,\ \exists\ output \mid \{S\} :=$ true

    for all $requests$ do
        Timer Start $\leftarrow$ $Current\ Time\ (\mu s)$
        $tryaccess \rightarrow UCS$
        Timer Stop $\leftarrow$ $Current\ Time\ (\mu s)$
    end for
```

From Figure 7(a) we observe that, when the number of attributes evaluated by the policy is equal or less than 5, the time required to perform the *pre-decision* phase is less than 20 milliseconds, while for 10 attributes the *pre-decision* phase takes about 22 milliseconds. Instead, from Figure 7(b) we see that, even considering a very large number of attributes, the time required to perform the *pre-decision* phase seems to be linear with respect to the number of attributes evaluated in the decision process.

We extended the results shown in Figure 7 by performing further experiments where the Usage Control service was loaded with a set of policies. In particular, Figure 8 reports the time required to perform the *pre-decision* process varying the total number of attributes when the Usage Control service enforces 1

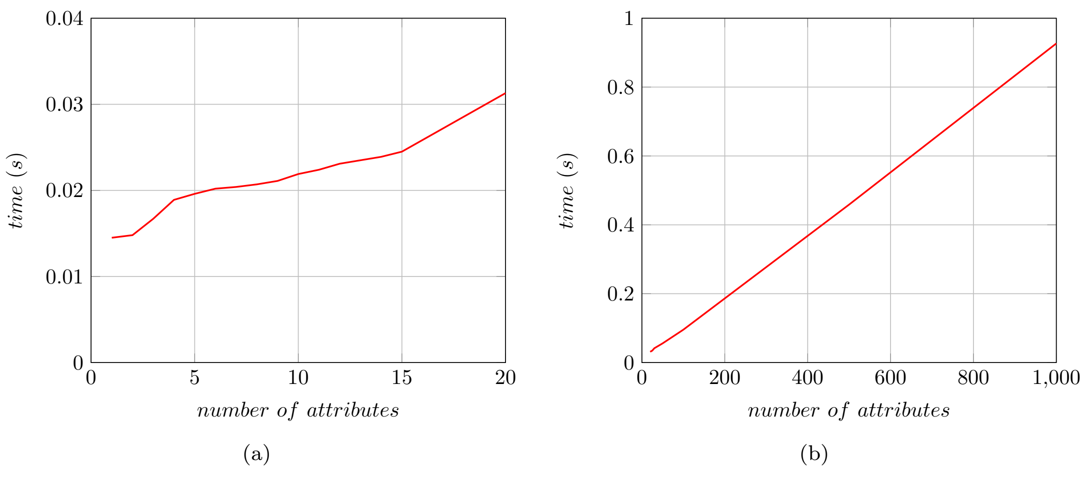


Figure 7: **Time required to perform the *pre-decision* phase with local Attribute Manager**

policy, 2 policies, 5 policies, and 10 policies. For each experiment, the attributes were distributes evenly among the policies, and the X axis reports the total number of attributes in the set of policies. For example, for 20 attributes, we measured the time required by the Usage Control service to evaluate one policy which contains 20 attributes, 2 policies with 10 attributes each, 5 policies with 4 attributes each, and 10 policies with 2 attributes each. Each experiment was performed 1,000 times, and we computed the average execution time. The variance was negligible.

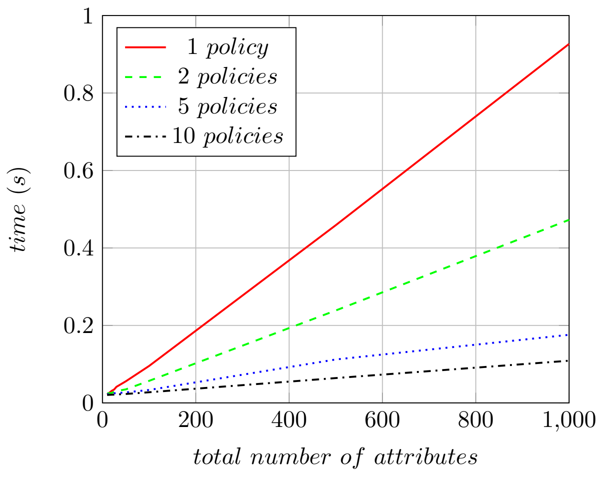


Figure 8: **Time required to perform the *pre-decision* phase with local Attribute Manager and multiple policies**

Figure 8 shows that, regardless of the total number of attributes, the execution time decreases when the number of policies increases. This is due to the optimization strategies implemented by the XACML engine we adopted in our framework, i.e., WSO2 Balana. In fact, in some cases, the WSO2 Balana XACML engine is able to return the result of the decision process by evaluating only a proper subset of the policies it receives as input, thus reducing the execution time. For instance, Figure 8 shows that, in case of 100 attributes, the time required to evaluate one policy (containing all the attributes) was about 95 milliseconds, the time for evaluating 2 policies (containing 50 attributes each) was about 57 milliseconds, while the time required to evaluate 10 policies (containing 10 attributes each) was about 28 milliseconds. The evaluation time decreases when the number of the policies loaded on the Usage Control service increases because we observed that the WSO2 Balana XACML engine was able to determine the result of the access request by evaluating only one of the input policies. In particular, since the combining algorithm adopted in our tests was "permit overrides" and the result of the evaluation of the first of the policies loaded on the system was “permit”, the WSO2 Balana XACML engine skipped the evaluation of the remaining policies. Hence, when the Usage Control service was loaded with 2 and 10 policies, the number of attributes actually evaluated to determine the access decision was, respectively, 50 and 10 out of 100. However, these results are highly dependent on the specific policies exploited for the performance tests. In fact, different policies could lead to different results. Summarizing, we observed that in our experiments the time required to perform the *pre-decision* process mainly depended on the number of policies and, consequently, of attributes that were actually taken into account in the decision process. Therefore, the results reported in Figure 7 represent the worst case from the performance point of view, because all the attributes in the policy were actually taken into account in the decision process.

Moreover, since we exploited a local AM, the delay due to the attribute retrieval phase is very low in the previous experiments. However, in some scenarios, remote AMs are exploited as well. In this case the time required to retrieve the updated attribute values is also affected by the delay introduced by the network, as described later.

The second set of experiments measured the time required by the Usage Control service to perform the decision processes related to the *Usage Control actions*: *tryaccess, startaccess* and *revokeaccess.* The tests were performed on the same testbed as before, varying the number of sessions. In particular, for a given user, a number N of sessions were created and started one by one (i.e., executing N times the *tryaccess* action and N times the *startaccess* action). The Usage Control policy enforced in our experiments includes a *pre-authorization* rule and an *on-authorization* rule that check that the value of the owner reputation, which is a mutable attribute, is equal to “excellent”. Hence, while the N sessions were in progress, we changed the value of this attribute to “bad”, and we measured the time required by the Usage Control service to decide which of the running sessions should be revoked. This time was measured starting from the moment when the PIP notified the CH of the new attribute value. The results are shown in Figure 9(a). Each experiment was performed 1,000 times and we computed the average of the results (the variance was negligible).

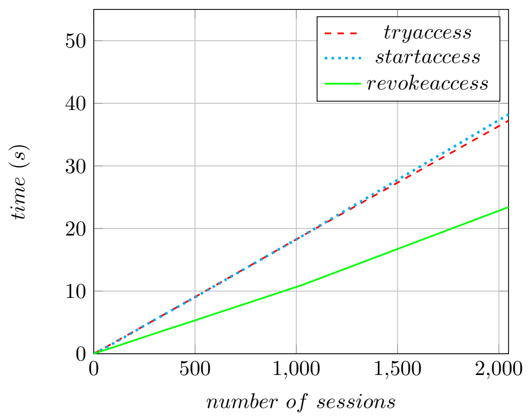
The pseudo-code we used to perform these measurements is described in Listing 5, and it was executed on the machine hosting the Usage Control service. Figure 9(b), instead, shows the normalized time, i.e., the total time divided by the number of sessions.

Listing 5 Pseudo-code for session based measurements

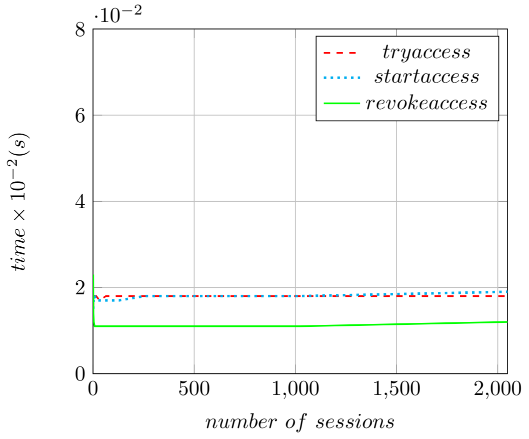
```
Require:
  $S : \{tryaccess,\ startaccess\}$
  $CONDITION : \forall\ requests,\ \exists\ output \mid \{S\} := \textbf{true}$
  $WHERE : tryaccess \prec startaccess$
  $STRING : u.reputation :=\ ''excellent''$

  for all $\{S\}$ do
    Timer Start $\leftarrow$ $Current\ Time\ (\mu s)$
    $\{S\} \rightarrow UCS$
    Timer Stop $\leftarrow$ $Current\ Time\ (\mu s)$
  end for
  $u.reputation \leftarrow\ ''bad''$
  Timer Start $\leftarrow$ $Current\ Time\ (\mu s)$
  $revokeaccess \rightarrow PEP$
  Timer Stop $\leftarrow$ $Current\ Time\ (\mu s)$
```

First of all, we noticed that the time required to perform the *tryaccess* action and the time required to perform the *startaccess* action are similar. This is because the two actions have very similar workflows, as detailed in Section 3.3, and because the *pre-policy* and the *on-policy* we used for our tests were similar. Moreover, we noticed that varying the number N of sessions that are created/started, the time required to perform the corresponding N *tryaccess*/*startaccess* actions is linear because, as shown by Figure 9(b), the time to perform one *tryaccess* or *startaccess* action does not depend on the number of existing sessions.

Concerning the *revokeaccess* action, we recall that the results shown in Figure 9(a) refer to the execution of one *revokeaccess* action only, which involves the re-evaluation of the *on-policy* for all the existing sessions. The total time required to re-evaluate the *on-policy* obviously depends on the number of sessions that need to be re-evaluated. However, Figure 9(b) confirms that the time required to re-evaluate the *on-policy* for a single session does not depend on the number of running sessions. Moreover, we noticed



(a) Total time



(b) Normalized time

	tryaccess(s)	*startaccess(s)*	*revokeaccess(s)*
1 Session	0.015	0.015	0.023
2 Sessions	0.033	0.035	0.028
4 Sessions	0.072	0.071	0.048
8 Sessions	0.147	0.141	0.089
16 Sessions	0.289	0.277	0.179
32 Sessions	0.565	0.529	0.342
64 Sessions	1.125	1.069	0.686
128 Sessions	2.302	2.202	1.354
256 Sessions	4.595	4.593	2.754
512 Sessions	9.228	9.245	5.454
1024 Sessions	18.717	18.707	10.918
2048 Sessions	37.223	38.221	23.422

(c) Reported values

Figure 9: **Time required to perform the *Usage Control* actions**

that the time required to decide to revoke N sessions is less than the time required to create or start them. This is due to the fact that, to create or to start N sessions, the *tryaccess* or the *startaccess* actions must be executed N times by the acPEP or by the startPEP, thus executing N communications with the Usage Control service. Instead, when the mutable attribute changes its value, the policy re-evaluation process is executed on all the N sessions, but only one *revokeaccess* message is sent to the revPEP (and hence one communication only is executed). This message reports the IDs of all the sessions that must be revoked. Finally, the time required to revoke a session measures how much time the Cloud resources have been used without holding the corresponding right. The results of our experiments show that, even in case of a very large number of sessions to be re-evaluated, this time is acceptable. As an example, the decision process for the revocation of 1024 session requires about 11 seconds.

In order to measure the delay introduced by our framework when the AMs exploited by the Usage Control service are remote, we performed further experiments. To this aim, we installed 5 MySQL servers supporting SSL connections on distinct machines located in our department network, and we configured 5 PIPs in the Usage Control service to exploit these MySQL servers as remote AMs. Each PIP performed a communication over SSL to retrieve the values of the attributes from the related AM. In particular, we fixed the total number of attributes evaluated by the Usage Control service, respectively, to 10 and 100, and we measured the time of the *pre-decision* phase when these attributes were allocated on 1, 2, and 5 remote AMs. We think that, in real scenarios, it is unlikely that a Cloud provider exploits more than a couple of remote AMs. Please notice that in our tests each PIP was configured to perform one query to retrieve all the required attributes from the related AM. The pseudo-code exploited to perform these measurements is shown in Listing 4, and it was executed on the machine running the Usage Control service. Each experiment was performed 1,000 times and we computed the average of the results (the variance was negligible).

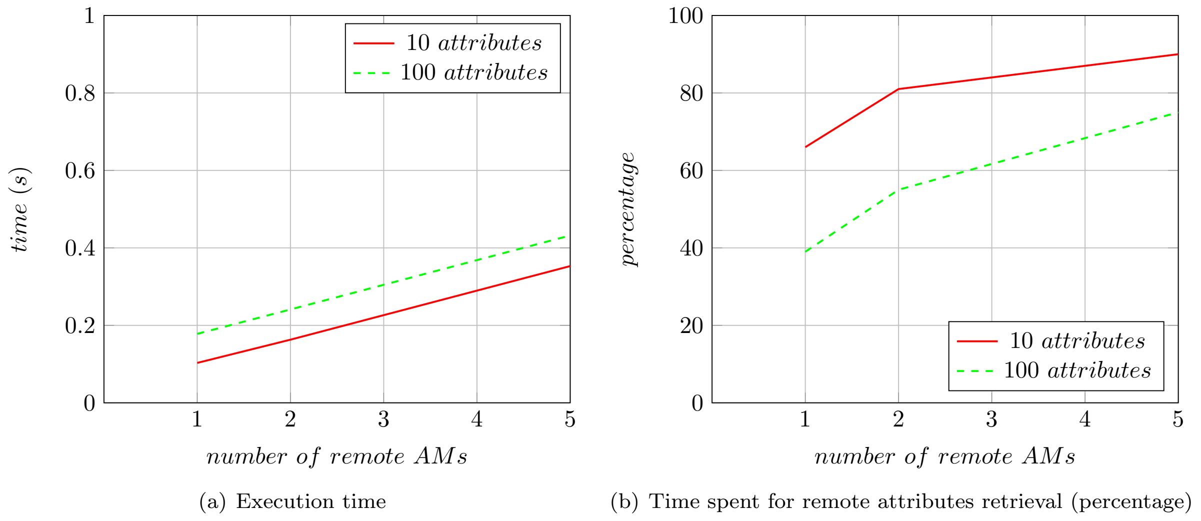


Figure 10: **Time required to perform the *pre-decision* phase with remote Attribute Managers**

Figure 10 reports the time required to perform the *pre-decision* phase varying the number of remote AMs, and shows the percentage of this time due to the retrieval of remote attributes. The results of the experiments where the total number of attributes is 10 (red line in Figure 10(a)) show that, in case of 1 remote AM (which stores 10 attributes) the time required to perform the *pre-decision* phase is about 100 milliseconds, in case of 2 remote AMs (which store 5 attribute each), this time is about 160 milliseconds, and when 5 remote AMs store 2 attributes each, the decision process takes about 350 milliseconds. The results of the experiments we performed with a Usage Control policy consisting of 100 attributes (green dashed line in Figure 10(a)) show that the evaluation time is about 180 milliseconds when all the 100 attributes are stored on one remote AM only, about 240 milliseconds when 2 remote AMs store 50 attributes each, and about 430 milliseconds when 5 remote AMs store 20 attributes each.

Comparing the results reported in Figure 10(a) with the ones in Figure 7, we observed that the adoption of remote AMs introduced an overhead which considerably affected the time required to perform the *pre-decision* phase. For example, the evaluation of a Usage Control policy with 10 attributes takes about 22 milliseconds when these attributes are local, about 100 milliseconds if these attributes are stored on one remote AM, and about 350 milliseconds in case of 5 remote AMs. In fact, the results in Figure 10(a) show that in our testbed the time required to retrieve remote attributes increases with the number of remote AMs that are exploited.

Moreover, from Figure 10(b) we observe that, for both 10 and 100 attributes, the remote attribute retrieval phase is the main factor which impacts on the performance of our framework. This overhead is mainly due to the communications between the PIPs and the AMs over the network, and depends on the latency of the network, and on the cost of secure connections. The adoption of AMs located in other domains with respect to the Usage Control service would lead to larger overheads. However, this time could be reduced in some scenarios by implementing proper strategies to optimize the communications. Our results are in line with the ones presented in [49], where the authors propose a Usage Control based security framework for collaborative computing systems, such as the Grid. In fact, the authors of [49] claim that, in their experiments, the time required by the Usage Control decision process was mainly due to the remote attribute retrieval phase.

The last set of experiments we performed on our testbed was aimed at evaluating the impact of the Usage Control service on the performance of OpenNebula based Cloud IaaS services. In other words, these experiments were meant to evaluate whether the adoption of the Usage Control service significantly affects the user experience when deploying a new VM on OpenNebula. Figure 11 shows the time required by OpenNebula to start a new VM both with and without the Usage Control service. Each test was performed 100 times, and Figure 11 reports the average of the results (the variance was negligible). As previously stated, in our testbed OpenNebula and the Usage Control service were deployed on distinct machines, and the communications between the PEP embedded in OpenNebula and the Usage Control service were over HTTPS. In these experiments, the Usage Control service evaluated 10 attributes which

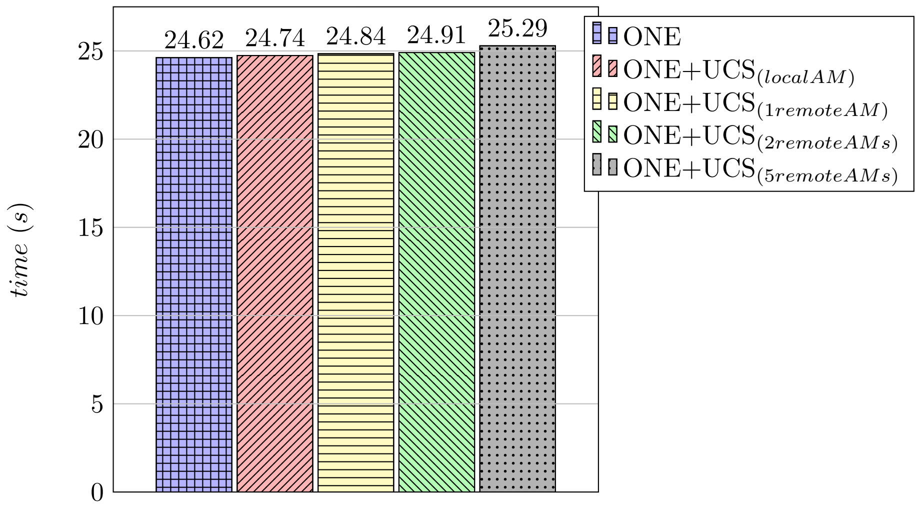


Figure 11: **Time required to deploy a Virtual Machine**

were allocated on a local AM or on a varying number of remote AMs. The operating system of the VM which was deployed on OpenNebula was Linux Ubuntu Desktop 14.04, and the VM image was downloaded from the OpenNebula Marketplace.

The results in Figure 11 show that, in case of local AM, the overhead due to the Usage Control service is about 0.5% of the VM deployment time. This overhead is mainly due to the secure communications between the PEP and the Usage Control service. When the attributes are provided by remote AMs, the overhead increases because of the attribute retrieval phase, which requires secure communications between the PIPs and the related AMs. In particular, our experiments show that, when the attributes are provided by one remote AM the overhead is about 0.9%, while exploiting 2 or 5 remote AMs the overhead is, respectively, about 1.2% and 2.7% of the VM deployment time. Hence, we think that, in our scenario, the overhead introduced by the Usage Control service cannot significantly affect the user experience.

6 Related Work

In our previous work we studied how to extend XACML to enforce UCON policies [10] and we provided a pilot implementation [25] for the Cloud environment. In this paper we provide the complete design and implementation of a framework for regulating the usage of Cloud IaaS services based on the Usage Control model, integrated within OpenNebula.

Gouglidis et al. [21] survey access control requirements for Cloud and Grid computing. They consider the UCON model as the best candidate to address these requirements. Another review of the security challenges of the Cloud environment has been published by Masood et al. [28]. Their paper presents a systematic analysis of the existing authorization solutions in Cloud and evaluate their effectiveness against well-established industrial standards that conform to the unique access control requirements in the domain. They also describe the adoption of UCON in Cloud systems using XACML.

Danwei et al. [13] and Tavizi et al. [41] propose the architecture for the enforcement of UCON policies in Cloud. The work of Danwei et al. [13] is focussed on the *pre-authorization* phase and on the privacy of security attributes. The authors handle these privacy issues by integrating trust negotiation in their model. The paper is a very high-level and does not provide details on the continuous control. The implementation of the proposed model is missing too. Tavizi et al. [41] focus mostly on the obligation enforcement. They argue that XACML should be extended to express UCON policies, but they do not provide details on how these policies can be enforced. Finally, it is not clear which Cloud systems can exploit their model.

The UCON model was successfully adopted in other distributed systems, e.g., the Grid [27, 49]. Sandhu et al. [49] consider which parts of the UCON model can be modelled exploiting standard XACML. In contrast, our approach considers how XACML should be extended to represent continuous control. Also, they require that attribute providers trigger the access re-evaluation, while we argue that the Usage Control service should be able to detect attribute changes through properly configured PIPs.

Recently, Yin et al. [45] proposed an extension of the XACML language for Usage Control, and they adopted it for regulating the usage of Cloud storage services in multi-Cloud platforms. Another work focused on Cloud storage services is [18]. In contrast, in our paper we focussed on IaaS services, with reference to OpenNebula. However, we think that our approach could be used to regulate the storage and sharing of data on the Cloud as well.

De Oliveira et al. [2] exploit obligations to define a new accountability policy language for the Cloud computing environment. Accountability policies are exploited by Cloud providers to protect the privacy of personal data.

Others interesting works are [16], which concerns e-health platforms for federated authorization, and [48] which extends the attribute based access control with CP-ABE algorithm [44].

A further interesting recent work is the one of Dan et al. [14], where they developed a centralized authentication and authorization architecture implementing several kind of access control models, although they didn't consider the Cloud computing environment.

Finally, a different approach is the one described in [12], where the Cloud provider enforces security policies by embedding an agent in each VM running in its Cloud.

7 Discussion

We remark that the proposed Usage Control authorization system, that runs as a separate service, is not very suitable to be exploited in case of very short lived operations, i.e., operations whose execution lasts a few milliseconds or less, such as disk accesses. In these cases the time required for the policy evaluation would be even greater than the time required for the operation itself. Hence, the delay introduced by the *pre-decision* phase would considerably affect the user experience by degrading the system performance. Moreover, the time required to perform the *on-decision* phase to detect a policy violation would be greater than the duration of the operation to be suspended. Consequently, the Usage Control service would not be effective, because it would try to suspend the operation when it is already terminated.

Nevertheless, we observe that the proposed framework can be applied in many different ways and to several other scenarios. We can envisage at least two dimensions that we are going to discuss further: *i)* the expressiveness of the Usage Control policies for continuous authorization (enhanced by the capability of exploiting multiple remote attribute providers); *ii)* the adoption of the Usage Control service to regulate the usage of other Cloud services. First of all, the Usage Control policies enforced by our framework go well beyond traditional access control ones, since they take into account the duration of the actions performed by the users and the mutability of attribute values over time. Indeed, the U-XACML language extends the expressiveness of XACML because Usage Control policies include authorizations, conditions and obligations which must be continuously satisfied for the whole duration of the actions. In other words, a policy violation causes the suspension of actions that are in progress. This is a notable advantage of our framework w.r.t. traditional access control ones, because the latter enforce their security policies only at access request time and no further controls are executed during the execution of the actions. Moreover, policies where actions performed by a user must be terminated as a consequence of other actions performed by other users can be expressed and enforced as well by our framework. Another advantage of the Usage Control service is the capability to exploit multiple providers of attributes, through the definition of proper PIPs that are able to interact with the AMs of these providers. In particular, besides expanding the set of attributes that a Cloud provider can exploit in his Usage Control policies, this features also allows to implement a sort of cooperation among the Usage Control services of distinct Cloud providers, by reading and updating the same attributes. As a consequence, our framework is able to suspend an action executed on a Cloud provider because of a policy violation caused by another action executed on another Cloud provider.

We think that our Usage Control service can be easily adopted to regulate the usage of resources in other types of Cloud services. For instance, we are currently working on its integration within Cloud Storage services implemented through the OpenStack framework, by suitably modifying the Swift component to embed our PEP. Moreover, we think that the Usage Control service could be successfully exploited also to regulate the usage of Cloud SaaS services, because the accesses to these services are long lasting as well. For instance, supposing that a Cloud SaaS provider offers a suite of office software to some companies, the employees of those companies will exploit this service for 8 hours every working day.

Summarizing, we believe that the applicability of the proposed service is thus quite significant and could really extend the flexibility of Cloud services for what concerns authorization mechanisms.

8 Conclusion

In this paper we described model, architecture and implementation of an enhanced authorization service based on the Usage Control model for regulating the usage of Cloud IaaS services. In particular, we showed that the proposed authorization service can be successfully adopted to deal with long lasting accesses, such as the execution of VMs. In fact, it is able to continuously enforce security policies while these accesses are in progress, and to suspend them when the corresponding rights do not hold any more. Besides the description of the implementation of the Usage Control based authorization system and of its integration within a IaaS Clouds service built on top of OpenNebula, this paper also presented a set of experimental results to evaluate the performance of the proposed solution. Concluding, we think that the integration of the Usage Control service within the OpenNebula toolkit for regulating the usage of the Cloud IaaS service represents a real improvement of the OpenNebula security support, allowing to continuously enforce more complex and fine grained security policies without significantly affecting the user experience.

9 Acknowledgements

This work was partially supported by the EU FP7 project *Confidential and Compliant Clouds* (Coco Cloud), GA #610853, and by the EU H2020 project *European Training Network for CyberSecurity* (NeCS), GA #675320.

References

[1] A. Alva, O. Caleff, G. Elkins, A. Lum, K. Pasley, S. Sudarsan, et al. The notorious nine: Cloud computing top threats in 2013. *Cloud Security Alliance*, 2013.

[2] M. Azraoui, K. Elkhiyaoui, M. Onen, K. Bernsmed, A. S. De Oliveira, and J. Sendor. A-PPL: An accountability policy language. In *Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance: 9th International Workshop, DPM 2014, 7th International Workshop, SETOP 2014, and 3rd International Workshop, QASA 2014, Revised Selected Papers*, volume 8872, page 319. Springer, 2015.

[3] M. Batet, K. Gibert, and A. Valls. The data abstraction layer as knowledge provider for a medical multi-agent system. In *Knowledge Management for Health Care Procedures*, pages 87–100. Springer, 2008.

[4] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

[5] R. Biswas and E. Ort. The java persistence api-a simpler programming model for entity persistence. *Sun, May*, 2006.

[6] G. Brunette, R. Mogull, et al. Security guidance for critical areas of focus in cloud computing v2. 1. *Cloud Security Alliance*, pages 1–76, 2009.

[7] R. Buyya, R. Ranjan, and R. N. Calheiros. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In *Algorithms and Architectures for Parallel Processing*, volume 6081 of *Lecture Notes in Computer Sciences* pages 13–31, 2010

[8] C. Caimi, C. Gambardella, M. Manea, M. Petrocchi, and D. Stella. Technical and legal perspectives in Data Sharing Agreements definition. In *Proceedings of the Annual Privacy Forum 2015 (APF15)*, volume 9484 of *Lecture Notes in Computer Sciences*, pages 178–192.

[9] D. Clark. Introducing eclipselink. *Eclipse Zone, Introducing EclipseLink/Eclipse Zone*, pages 1–3, 2008.

[10] M. Colombo, A. Lazouski, F. Martinelli, and P. Mori. A proposal on enhancing XACML with continuous usage control features. In *proceedings of CoreGRID ERCIM Working Group Workshop on Grids, P2P and Services Computing*, pages 133–146. Springer US, 2010.

[11] M. Coppola, P. Dazzi, A. Lazouski, F. Martinelli, P. Mori, J. Jensen, I. Johnson, and P. Kershaw. The CONTRAIL approach to cloud federations. In *Proceedings of The International Symposium on Grids and Clouds (ISGC 2012), Proceedings of Science*, 2012.

[12] J. Daniel, F. El-Moussa, G. Ducatel, P. Pawar, A. Sajjad, R. Rowlingson, and T. Dimitrakos. Integrating security services in cloud service stores. In *Trust Management IX. IFIP Advances in Information and Communication Technology*, volume 454, pages 226–239. Springer International Publishing, 2015.

[13] C. Danwei, H. Xiuli, and R. Xunyi. Access control of cloud service based on UCON. In *Proceedings of the 1st International Conference on Cloud Computing*, pages 559–564. Springer-Verlag, 2009.

[14] P. Das and A. Das. Centralized authorization service (CAuthS) or authorization as a service (AuthaaS) - A conceptual architecture. *International Journal of Computer Applications*, 113(18), 2015.

[15] D. Dave, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile. RFC5280, 2008.

[16] M. Decat, D. Van Landuyt, B. Lagaisse, and W. Joosen. On the need for federated authorization in cross-organizational e-health platforms. In *Proceedings of the 8the international conference on Health Informatics*, volume 8, pages 540–546, 2015.

[17] M. D. Dikaiakos, D. Katsaros, P. Mehra, G. Pallis, and A. Vakali. Cloud computing: distributed internet computing for it and scientific research. *Internet Computing, IEEE*, 13(5):10–13, 2009.

[18] T. Esther Dyana and S. Maheswari A Secure Data Storage and Trustworthy Resource Sharing In Cloud Computing Environment. *International Journal of Advances in Computer Science and Technology*, 4(4), 2015.

[19] D.-G. Feng, M. Zhang, Y. Zhang, and Z. Xu. Study on cloud computing security. *Journal of software*, 22(1):71–83, 2011.

[20] R. Gellman. Privacy in the clouds: risks to privacy and confidentiality from cloud computing. In *Proceedings of the World privacy forum,*, 2012.

[21] A. Gouglidis and I. Mavridis. On the definition of access control requirements for grid and cloud computing systems. In *Networks for Grid Applications*, volume 25 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 19–26. Springer Berlin Heidelberg, 2010.

[22] K. Hamlen, M. Kantarcioglu, L. Khan, and B. Thuraisingham. Security issues for cloud computing. *Optimizing Information Security and Advancing Privacy Assurance: New Technologies: New Technologies*, page 150, 2012.

[23] B. Katt, X. Zhang, R. Breu, M. Hafner, and J. Seifert. A general obligation model and continuity: enhanced policy enforcement engine for usage control. In *Proceedings of the 13th ACM symposium on Access control models and technologies (SACMAT '08)*, pages 123–132, 2008.

[24] R. Kumar, N. Gupta, S. Charu, K. Jain, and S. K. Jangir. Open source solution for cloud computing platform using openstack. *International Journal of Computer Science and Mobile Computing*, 3(5):89–98, 2014.

[25] A. Lazouski, G. Mancini, F. Martinelli, and P. Mori. Usage control in cloud systems. In *International Conference for Internet Technology And Secured Transactions, 2012*, pages 202–207. IEEE, 2012.

[26] A. Lazouski, F. Martinelli, and P. Mori. Usage control in computer security: A survey. *Computer Science Review*, 4(2):81–99, 2010.

[27] F. Martinelli and P. Mori:. On usage control for grid systems. *Future Generation Computer Systems*, 26(7):1032–1042, 2010.

[28] R. Masood, M. A. Shibli, Y. Ghazi, A. Kanwal, and A. Ali. Cloud authorization: exploring techniques and approach towards effective access control framework. *Frontiers of Computer Science*, pages 1–25, 2015.

[29] P. Mell and T. Grance. The NIST definition of cloud computing. recommendation of the national institute of standards and technology. Technical report, NIST, 2011.

[30] D. Milojičić, I. M. Llorente, and R. S. Montero. Opennebula: A cloud management tool. *IEEE Internet Computing*, 15(2):0011–14, 2011.

[31] R. Neisse, A. Pretschner, and V. Di Giacomo. A trustworthy usage control enforcement framework. *International Journal of Mobile Computing and Multimedia Communications*, 5(3):34–49, 2013.

[32] E. Network and I. S. Agency. *Cloud Computing: Benefits, Risks and Recommendations for Information Security.* ENISA, 2009.

[33] OASIS. Oasis eXtensible Access Control Markup Language (XACML) version 3, OASIS standard. Technical report, 22 January, 2013.

[34] M. Ozsu and P. Valduriez. *Principles of Distributed Database Systems (3rd ed.).* Springer Science, LLC, 2011.

[35] J. Park and R. Sandhu. The UCON$_{ABC}$ usage control model. *ACM Transactions on Information and System Security*, 7(1):128–174, 2004.

[36] A. Pretschner, M. Hilty, and D. A. Basin. Distributed usage control. *Communications of the ACM*, 49(9):39–44, 2006.

[37] B. P. Rimal, E. Choi, and I. Lumb. A taxonomy and survey of cloud computing systems. In *INC, IMS and IDC, 2009. NCM'09. Fifth International Joint Conference on*, pages 44–51. IEEE, 2009.

[38] S. Subashini and V. Kavitha. A survey on security issues in service delivery models of cloud computing. *Journal of network and computer applications*, 34(1):1–11, 2011.

[39] H. Takabi, J. B. Joshi, and G.-J. Ahn. Security and privacy challenges in cloud computing environments. *IEEE Security & Privacy*, 8(6):24–31, 2010.

[40] B. Tang and R. Sandhu. Extending openstack access control with domain trust. In *Network and System Security*, pages 54–69. Springer, 2014.

[41] T. Tavizi, M. Shajari, and P. Dodangeh. A usage control based architecture for cloud environments. In *IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, pages 1534 –1539, 2012.

[42] G. Toraldo. *OpenNebula 3 Cloud Computing.* Packt Publishing Ltd, 2012.

[43] L.M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2008

[44] B. Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *Public Key Cryptography–PKC 2011*, pages 53–70. Springer, 2011.

[45] K. Z. Yin and H. H. Wang. Mcacm: A cloud storage access control model for multi-clouds environment based on xacml. In *Applied Mechanics and Materials*, volume 713, pages 2451–2454. Trans Tech Publ, 2015.

[46] Y. A. Younis, K. Kifayat, and M. Merabti. An access control model for cloud computing. *Journal of Information Security and Applications*, 19(1):45–60, 2014.

[47] K. Zeilenga. Lightweight directory access protocol (ldap): Technical specification road map. RFC4510, 2006.

[48] L-x. Zhang and J-s. Zou. Research of ABAC mechanism based on the improved encryption algorithm under cloud environment. In *Ubiquitous Computing Application and Wireless Sensor*, pages 463–469. Springer, 2015.

[49] X. Zhang, M. Nakae, M. J. Covington, and R. Sandhu. Toward a usage-based security framework for collaborative computing systems. *ACM Transactions on Information and System Security*, 11(1):1–36, 2008.

[50] X. Zhang, F. Parisi-Presicce, R. Sandhu, and J. Park. Formal model and policy specification of usage control. *ACM Transactions on Information and System Security*, 8(4):351–387, 2005.

[51] D. Zissis and D. Lekkas. Addressing cloud computing security issues. *Future Generation Computer Systems*, 28(3):583–592, 2012.