Architecture, Workflows, and Prototype for Stateful Data Usage Control in Cloud

Aliaksandr Lazouski, Gaetano Mancini, Fabio Martinelli, Paolo Mori
Istituto di Informatica e Telematica, Consiglio Nazionale delle Ricerche
Pisa - Italy
e-mail: {aliaksandr.lazouski,gaetano.mancini, fabio.martinelli,paolo.mori}@iit.cnr.it

***Abstract*—This paper deals with the problem of continuous usage control of multiple copies of data objects in distributed systems. This work defines an architecture, a set of workflows, a set of policies and an implementation for the distributed enforcement. The policies, besides including access and usage rules, also specify the parties that will be involved in the decision process. Indeed, the enforcement requires collaboration of several entities because the access decision might be evaluated on one site, enforced on another, and the attributes needed for the policy evaluation might be stored in many distributed locations.**

***Index Terms*—Usage Control, UCON, Cloud System, Attributes, Concurrency Control**

I. Introduction

Controlled usage of data objects is a challenging issue. When data are stored in the domain of the originator, traditional access control techniques can be adopted to check that users hold the proper access rights. Once data have been copied and/or moved to another domain, typically no further controls are performed to regulate their subsequent usage. However, many scenarios require that the usage of data is regulated even when they are not stored in the originator's domain any more. Controlling the usage of data that have been stored outside the originator's domain involves a large number of challenging issues.

We exemplify our approach by considering Virtual Machine Images (VMIs). Consider a user U that creates a VMI I, that is provided by a Cloud IaaS service, to other users. U could want that his image is used by users as long as their reputation is above a given threshold. Hence, as soon as the reputation of the user goes under the threshold, his virtual machine running I should be suspended or interrupted.

To overcome limitations of the traditional access control models and address the previous needs, usage control models [5] can be adopted, either the UCON approach of Park and Sandhu [3], [4] which focuses mainly on continuity of authorization and mutability of attributes or distributed usage control [10] that focuses more on control of disclosed data.

This paper describes a framework of usage control for enforcing policies on data objects stored in distributed systems that is an extension of the UCON model of Sandhu. The policy, beside defining the rules that must be satisfied before and during the access, also regulates other aspects of the decision process. In particular, it allows the authorization system to find which Policy Decision Points (PDPs) and Policy Information Points (PIPs) are trusted to evaluate the policy itself. The paper also presents initial implementation that enforces such policies in distributed systems, addressing the problem of the concurrent accesses to attributes. Indeed, concurrency issues are very crucial in usage control. The UCON model introduces updates of attributes as a result of the access evaluation. Thus, two or more decision processes can read or update the same attribute simultaneously. It is important to avoid concurrency collisions which may lead to severe breaches in security. In fact, the access may be granted to unauthorized users and denied for legitimate ones. Hence, interleaving between concurrent decision processes should be controlled.

As mentioned, we have chosen Cloud IaaS services as the reference scenario. In particular, VMIs, provided by a Cloud IaaS service, are data objects which are created by some users but stored outside their domains and used by other users of the service. For simplicity, the reference scenario assumes that data objects are always stored in trusted domains, i.e., Cloud service providers, in order to guarantee that the right security policy is always and promptly enforced. Hence, in this paper we focus on the definition of a policy and of the distributed architecture for the policy evaluation and enforcement, including resolution of concurrency issues. We leave for future work the scenario where data are stored in domains that are not trusted or where the network connection could be down for some time, or where some components of the framework might crash.

1) Motivation and Contribution: There are many scenarios that require controlling the usage of data objects stored in distributed systems outside the originator's domain. For instance, e-health scenarios, where medical documents (e.g., examination results or prescriptions) must be protected when are sent outside the domain of a hospital that produces them, or the distribution of Multimedia Contents, or controlling VMIs that are executed in Cloud IaaS services.

An important improvement of the proposed approach with respect to Digital Right Management is that the usage control model proposes complex usage policies where rights to access

This work was supported by the FP7 projects *Open Computing Infrastructures for Elastic Services* (CONTRAIL), *Network of Excellence on Engineering Secure Future Internet Software Services and Systems* (NESSOS), and *Confidential and Compliant Clouds* (CoCoCloud)

DOI 10.1109/SPW.2014.13

and to use data depend on many factors, that could be independent from the data that are being accessed, such as subject's features, or even the features of other data objects. Also, the usage control model is very expressive and it defines policies that, beside being used to decide the rights of execution of an action on the data, are enforced the whole time the action is in progress. Consequently, the action on the data could be interrupted while in progress because the update of some attributes lead to the policy violation.

Finally, this paper defines the reference architecture and its initial implementation for enforcement of usage control policies in distributed systems when data are stored outside the originator's domain. This architecture allows exploitation of several PDPs and PIPs performing the decision process, and it also addresses the concurrency issues that could arise when several decision processes involving the same attributes are executing concurrently.

This paper is organised as follows. We define the reference scenario in Section II. We introduce our extension to the UCON model in Section III. Section IV describes the architecture, the workflow. Section V introduces the initial implementation of the proposed model. We finish the article with related work (section VI) and conclusions (section VII).

II. Reference Scenario

The reference scenario concerns the Cloud environment, where VMIs provided by a Cloud IaaS service represent data objects that are created by some users but accessed by others.

For example, the NESSoS EU project[a] produced a VMI where a set of security tools, provided by several partners of the project, had been properly installed and configured, and they are ready to be used. This VMI was uploaded on the NESSoS Cloud Execution Environment, that is a cluster of workstations running OpenNebula[b] to provide the IaaS service to NESSoS users. The NESSoS administrators define a security policy that determines which actions can be performed on the VMI and under which conditions. The Cloud IaaS service provides the trusted domain where the VMI can be stored and run in accordance with the policy.

This VMI is used by NESSoS users, who run it creating a new Virtual Machine (VM), i.e., performing a long-lasting $execute$ action. Distinct users can create their distinct VMs from the same VMI. The $execute$ action ends when the user stops the VM that he created.

The user can ask the Cloud IaaS service to replicate the VMI ($replicate\&store$ action), i.e., to take a snapshot of the running VM and to create a new VMI, in order to preserve some updates he did on the VMI. This new VMI can be stored in the Cloud IaaS service where also the original VMI is stored, or it could be moved (through the $migrate$ action) to another Cloud. The new VMI has the same security policy as the original VMI. The $replicate\&store$ action is also long-lasting and it ends when the new VMI is destroyed.

[a]http://www.nessos-project.eu

[b]http://www.opennebula.org

III. Extending Usage Control Model with Multiple Policies

A data object can be stored at any site in a distributed system and can be passed across different security domains. Wherever the data object resides in the system, access to it and its usage should be regulated according to the usage control policy which is defined by the data originator.

The usage control policy is based on the Usage Control model (UCON), that has been defined by Sandhu et al. in [3]–[5]. The usage control policy states that some accesses to the data object are long-lasting and must be controlled continuously. As soon as the policy does not hold any more, these accesses should be interrupted.

In distributed settings the access decision might be evaluated on one site, enforced on another, and attributes needed for the policy evaluation might be stored in many locations. We extends the UCON model by addressing these issues.

A. Components

The core components of the UCON model are: subjects, objects, actions, attributes, access decision factors (authorizations, conditions, obligations). Here we describe how subjects, objects, actions, and attributes are instantiated in our approach.

Subjects, Objects and Actions.

The main actors of our scenario are:

- the *Data Originator* (DO) that is the entity that produces and keeps the *Original Data* (OD);
- the *Data User* (DU), a subject, that can access the OD, derive from the OD a similar (or identical) object called *data copy* (DC), and move the DC to other DUs.

Fig. 1 describes actions defined on ODs and DCs and their durability in time:

- $replicate\&store(s, o_{od}, o_{dc})$: the DU ($s$) calls it when he wants a new DC (o_{dc}) derived from the OD (o_{od});
- $migrate(s_{from}, s_{to}, o_{dc})$: the DU calls this action when he wants to move the DC (o_{dc}) to another DU (s_{to});
- other actions related to the specific data object. Supposing that the data object is a VMI, a possible action could be:
  - $execute(s, o_{dc})$: the DU calls this action to run the VM exploiting the image o_{dc}. The action terminates when the VM is stopped.

Access to ODs and DCs is regulated by a data usage policy that is embedded into data, i.e., the policy migrates with DCs. The policy is detailed in subsection III-B.

Attributes. Attributes describe subjects' and objects' features. Attributes can be mutable. Mutable attributes can be updated before (*pre-update*), during (*on-update*), or after (*post-update*) the access.

The nature of attributes is diverse. For instance, an attribute can refer to: (i) a specific DC (e.g., "id" or "creation date" of a VMI), (ii) a subset of DCs (e.g., "number of VMIs belonging to the user U and stored in the Cloud C"), (iii) all DCs derived from the OD (e.g., "global number of VMIs").

Attribute values can be modified as a result of: (i) the access decision process ("global number of VMIs" is changed

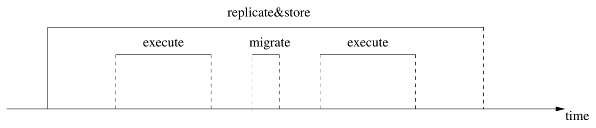

Fig. 1. Actions on VMI

before and after execution of the $replicate\&store$ action), (ii) other reasons, e.g., administrative actions, deliberate actions of a user (e.g., the attribute "user role" is modified by the administrator when the user gets an advancement, the attribute "user location" is modified when the user moves from one place to another). Some attributes can be included in both categories. For instance, the attribute "balance" of the user can be decreased as a result of the access (the user must pay for running a VM), or increased by the user via a bank transaction.

In order to perform a decision process efficiently, it is crucial to decide where various attributes are stored. Immutable attributes can be embedded in a data object (e.g., attributes like "id", "creation date", "originator") because it is unlikely that their values change during the object lifetime.

Mutable attributes, instead, change their values frequently. Hence, embedding them in data objects could lead to a large overhead for keeping consistency among all copies of the same attribute. Consequently, we prefer to have only one copy of each attribute and to store it in a specific location which is defined by PIP allocation policy (see subsection III-B3). The unique copy of the attribute might be accessed and modified by many decision processes which run concurrently and subsection IV-B1 covers issues related to the concurrency control of mutable attributes.

B. Data Usage Policy

Data usage policy paired with each data object consists of a set of auxiliary sub-policies, that regulate distinct aspects of the decision process. In fact, besides the Usage Control Policy, that defines the right of users to access objects, we introduce two kinds of allocation policies, PDP Allocation Policy and PIP Allocation Policy, which complement all the aspects required to set up the policy enforcement.

1) Usage Control Policy: determines whether a subject holds proper rights on a data object, and it is enforced while the access in progress. We present a simple example of Usage Control Policyfor our reference scenario. However, this policy is expressed using a human-readable language:

```
policy-1:
 action:replicate&store(s, o, o')
 pre-authorization:(s.nCopyStored+s.nCopyMigrated<X)
                   AND (s.role==GOLDUSER)
 pre-update:(s.nCopyStored++) AND
            (o'.createdBy:=s.id) AND
            (o'.isCopyOf:=o.id) AND
            (o'.storedCountry:=o.storedCountry)
 on-authorization:(o'.storedCountry==o.storedCountry)
 post-update:(s.nCopyStored--)
policy-2:
 action:execute(s, o)
 pre-authorization:(s.nRunning<Y) AND
                   (o.storedCountry==Italy)
 pre-update:(s.nRunning++)
 on-authorization:(s.reputation>T)
 post-update:(s.nRunning--)
```

The policy labelled policy-1 regulates the replication of VMIs. The pre-authorization predicate states that the user s can create a new data object only if the total number of objects belonging to him is less than X and his role is GOLDUSER. The pre-updates initialize attributes of the new object. For example, the attribute storedCountry is initialized with the id of the country where the OD is stored. The on-authorization predicate states that the object must reside in the same country as the OD for its lifetime. Hence, if the new data object is migrated to another country, the policy states that the right of storing that object expires, and the access must be revoked.

The second policy concerns execution of existing VMIs. The pre-authorization predicate states that a user can execute a further VM if he is currently executing less than Y VMs, and if the VM is running in Italy. The pre-update increases the number of running VMs belonging to the user. The on-authorization predicate requires that the reputation of the user that runs the VM is greater than the threshold T during the VM lifetime. Hence, if the reputation of the user goes under the threshold T, the running VM must be interrupted.

2) PDP Allocation Policy: is evaluated to determine which PDP is trusted to evaluate the Usage Control Policy. We model the PDP Allocation Policy as a mapping between a policy id and a PDP location which can be *local*, *global*, or a third party PDP. The local PDP is placed on a client side, and the global PDP is placed on a data originator site. The third party PDP resides in a remote domain that is trusted by the data originator. Besides the level of trust, the performance could be another factor that is taken into account when defining the PDP Allocation Policy.

An example of PDP Allocation Policy is the following:

```
policy-1 (replicate&store) : <global_PDP_URL>,
                             <remote_PDP1_URL>,
                             <remote_PDP2_URL>
policy-2 (execute) : <local_PDP_URL>
```

3) PIP Allocation Policy: is evaluated to find PIPs for retrieving fresh values of attributes required to carry out the decision process, i.e., to evaluate the Usage Control Policy. The PIP Allocation Policy specifies addresses of the PIPs to which attribute queries must be sent. The selected PIPs must be able to interact with Attribute Managers (AMs) that provide raw values of attributes.

When the PDP needs an attribute, either it finds it in the access request, or the PDP queries the right PIP that knows which AM is in charge of managing the required attribute and which is the protocol to communicate with this AM.

An example of PIP Allocation Policy is the following:

```
nCopyStored,nCopyMigrated,nRunning:<remote_PIP1_URL>
role:<remote_PIP2_URL>
reputation:<remote_PIP3_URL>
createdBy,isCopyOf,storedCountry:<local_PIP_URL>
```

IV. Architecture

The architecture of our system, shown in Figure 2, extends the common authorization architecture [9] to deal with a continuous policy enforcement in distributed environment. The architecture is distributed because the components are spread

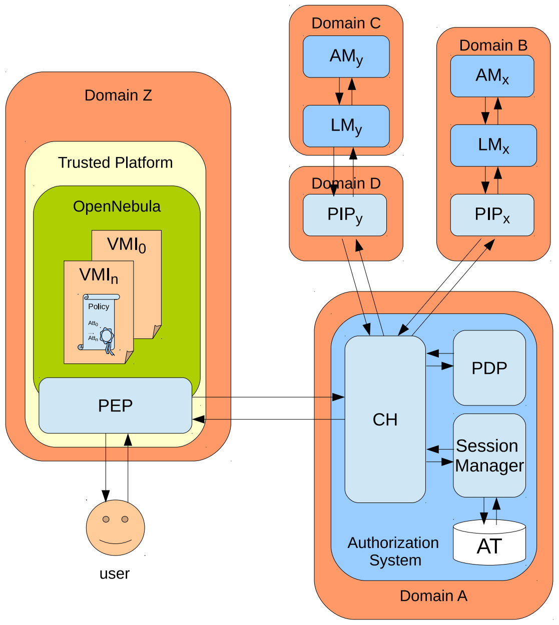

Fig. 2. Architecture for Enforcement of Data Usage Policy

over several domains, and it has been designed to deal with concurrent retrieval and update of mutable attributes.

We assume that the domain were the data are stored (i.e., domain Z in Figure 2) is trusted, and hence we don't propose any solution for the secure storage for these data and to ensure that the data usage policy is always enforced, leaving these issues as future work. Also, the domains that host the authorization system and the services for attribute management are trusted by the DO, because they have been chosen according to the allocation policies which the DO embedded in his data.

A. Components

The main components of the architecture are the following:

- Policy Decision Point (PDP) evaluates usage control policies and produces the access decision;
- Policy Enforcement Point (PEP) intercepts invocations of access requests, suspends them before starting, chooses PDPs that will be exploited for the decision processes according to PDP Allocation Policy, queries the PDPs for access decisions, enforces the obtained decisions by resuming suspended requests, and interrupts ongoing accesses when the policy violation occurs;
- Context Handler (CH) is the front-end of the authorization system, that manages the protocol for communicating with PEPs, PDPs, and PIPs. It converts and forwards messages sent between components in the proper format.
- Attribute Manager (AM) manages attributes and knows their current values;
- Policy Information Point (PIP) provides interface for other components to query attributes. The PIP communicates then with a specific AM to retrieve attributes;
- Lock manager (LM) determines whether an attribute query/update should be passed for execution to the AM, or should be delayed by placing it in a queue and executing it later. The LM might be embedded into the AM;
- Session Manager (SM) is responsible for continuous control and manages ongoing accesses, i.e., usage sessions;
- Access Table (AT) keeps meta-data regarding usage sessions. It contains a table of the current sessions with their statuses and the table of IDs of attributes needed to service each session.

All components are crucial for the correct enforcement of a data usage policy, and hence they should be placed in domains that are trusted by the DOs.

For what concerns the PDPs and the PIPs, the aim of the PDP Allocation Policy and of the PIP Allocation Policy is exactly to locate authorization system components which are considered as trusted by the DO, and hence can be exploited for the execution of the decision process. The PEP, instead, should be located where the data are stored, in order to be able to intercept all the security-relevant access requests related to these data. Since the PEP must be non-bypassable and tamper-proof, it must be placed in a trusted component (or machine) that guarantees these features.

In our reference example the PEPs must be installed on the Cloud provider site, that represents a trusted domain for the DO. In particular, the PEPs should be integrated within the framework that implements the Cloud IaaS service (e.g., OpenNebula or OpenStack), that is the component that really executes the actions on the VMIs. Most of the available frameworks are configurable to exploit external and customized authorization systems.

B. Workflows

1) Concurrent Retrieval and Update of Attributes: The policies paired with data objects may involve a common subset of attributes and, consequently, some concurrency issues in attribute management arise when several policy decision processes are performing concurrently.

For example, the policy-1 from our reference scenario allows the creation of a new data object only if the total number of data objects belonging to a user s is less than X. Let us suppose that s tries to create two distinct replicas of the object at the same time. This triggers two policy decision processes. Each policy decision process follows these operations: *1)* retrieves the total number of data objects belonging to s; *2)* compares the value of the attribute with X; and *3)* if the current value of the attribute is less than X, increases the attribute `s.nCopyStored` by 1 and allows creation of a new data object.

A concurrency problem, called *race condition*, arises when the resulting access decisions depend on interleaving of operations performed by these decision processes. *Lost update* and *inconsistent retrieval* are examples of the race condition problem. Lost update happens when all decision processes read the old value of the attribute simultaneously while decide on and update the new attribute value independently. Assume that

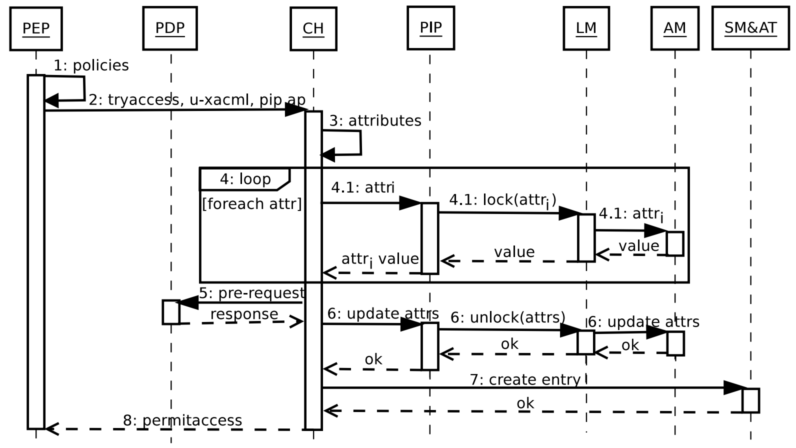

Fig. 3. Sequence Diagram for Pre-Authorization

the second decision process retrieves the value of the same attribute after the first process performed the operation 1 but before the execution of the operation 3. As a matter of fact, if the current value of the total numbers of data objects belonging to s is N-1, both decision processes will allow the creation of new data objects, while only one data object can be created according to the policy. Inconsistent retrieval occurs when a policy decision process retrieves one attribute before another decision process updates it, and reads another attribute after the same decision process has updated it.

The two-phase locking protocol (2PL) is the solution we choose to address concurrency issues in attribute retrieval and updates. Due to space limitation we refer to [8], [15] where the family of these protocols is described in details.

Two-Phase Locking. To avoid concurrency issues, each attribute is paired with a lock, and any operation on the attribute value requires acquisition of the lock. When the decision process requires to read and update one attribute or more, all these attributes are locked before performing the operations. If the attribute is already locked, i.e., there is another concurrent decision process (or more) that involves the same attribute, the decision process is suspended until the lock is released by the other decision process. As soon as the decision process has been performed, the involved attributes are unlocked.

To avoid deadlocks, the policy decision process must lock attributes in a predefined order. The order of unlocking is arbitrary. We assume that it is possible to define the attribute ordering because a set of attributes needed for enforcement of a usage control policy is known a priori and this set is specified in the PIP allocation policy. If the policy requires same attributes of different principals, attributes are ordered based on the holders' ids.

2) Pre-authorization: Basic Workflow Steps: (1) The PEP intercepts the access request T_i performed by a user, determines the subject and the object of T_i and gets the Usage Control, PDP and PIP allocation policies from the object; (2) The PEP enforces the PDP allocation policy and determines which PDP should perform the decision process to authorize T_i. The PEP attaches the usage control and the PIP allocation policies to T_i and sends everything to the CH of the chosen PDP; (3) The CH initiates the retrieving of attributes. The CH determines all attributes needed to evaluate the usage control policy. For each attribute, the CH determines the related PIP by enforcing the PIP allocation policy; (4) The CH starts pulling attributes according to the selected concurrency control algorithm (e.g., a two-phase locking with attributes queried in the predefined order). For each attribute $attr_k$, the CH sends to the corresponding PIP$_k$ the attribute query. The CH awaits until all attributes are collected; (4.1) The PIP$_k$ receives the request from the CH for the attribute $attr_k$ and forwards it to LM$_k$ which tests if $attr_k$ is locked. If so, LM$_k$ delays the attribute query. Otherwise, LM$_k$ sets a lock on $attr_k$, and then sends the attribute query to the AM. The AM provides the value and the LM$_k$ replays back to the CH via the PIP$_k$. The lock is not released yet; (5) The CH sends T_i, collected attributes, and the usage control policy to the PDP for the access evaluation. The PDP evaluates the usage control policy and replies to the CH with the access decision and the new values of attributes; (6) The CH sends the new attribute values to the PIPs. Since attributes are still locked because of step 4.1, the LMs simply ask the AMs to update them. Then, the LMs release locks for attributes used for T_i. From now on, the attributes are available for other accesses; (7) When all attributes are updated, the CH is ready to reply to the PEP with the access decision. If the access decision is "permit", the CH sends the *create entry* message to the SM for creating a new entry in the AT that represents the new usage session. (8) Finally, the CH replies with the access decision to the PEP; (9) The PEP enforces the access decision allowing or denying the execution of T_i.

3) Continuous Control: If the pre-authorization phase allows the execution of T_i, the continuous control phase is started. The workflow is as follows: (10) when the access to the data object has began (e.g., the VM is running in our reference example), the PEP sends the notification to the CH

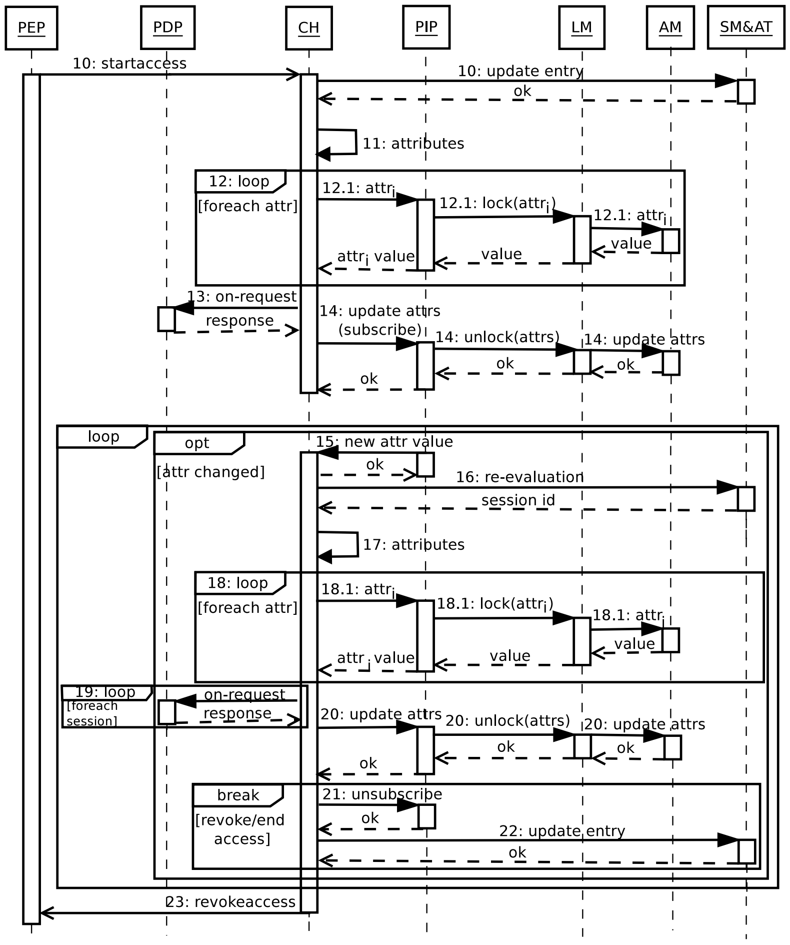

Fig. 4. Sequence Diagram for Ongoing Authorization

that, in turn, forwards it to the SM. The SM contacts the AT to change the status field of the database entry related to this usage session indicating that the session is now active; (11-13) Then, the continuous control phase starts, and the CH behaves as in steps 3-5 of the pre-authorization phase, i.e., it retrieves the current values of the attributes (locking them), it asks the PDP to evaluate the UCON ongoing policy, and it receives the decision along with the attribute updates to be performed; (14) The CH sends the new attribute values to the PIPs. Assume, the access decision produced by the PDP in steps 11-13 is “permit”, then the CH, along with new values, sends the *subscription* requests for the attributes to the PIPs. The PIPs contact the related LMs, and since attributes are already locked because of the previous step, the LMs simply ask the AMs to update them. Then, the LMs release locks for attributes used to service the access request. From this moment on, the attributes are available for other accesses; (15) From then on, the authorization system idles. When a value of any subscribed attribute is changed, the CH receives a notification from the PEP due to the previous subscription. The notification triggers the access re-evaluation. The CH sends the attribute name to the SM; (16) The SM replies with a set of active sessions whose policies involve this attribute; (17) The CH determines a set of attributes which should be refreshed and which are needed for the sessions re-evaluation; (18) Then, the CH behaves as the steps 4 of the pre-authorization phase. Attributes are locked by the CH for the PDP that re-evaluates the local sessions, while other PDPs do wait unless they are unlocked. Therefore, there is only one PDP at a time in the system which possesses the lock over attributes; (19) In order to perform the local re-evaluation of all active sessions, the CH stores in a local cache the values of the collected attributes. For each active session which should be re-evaluated, the CH determines a subset of needed attributes and sends them with the usage control policy to the PDP for the access re-evaluation. The order in which active sessions are re-evaluated could affect decision processes. How N ongoing sessions should be re-evaluated (in what order) upon receiving a new attribute value is out of scope of this paper. Furthermore, the re-evaluation of the policy for a given session may update an attribute value which might triggers the re-evaluation of some other local sessions. This could lead to possible endless loops. This is an intrinsic UCON issue, and to address it we need a formal model of attributes, policies, and update actions. In this paper, we do not focus on it. Simple solution may require that sets of attributes used in ongoing authorization predicates and ongoing update actions do not intersect; (20) When all sessions are re-evaluated, the CH takes the updated attribute values from the local cache and sends new attribute values to the related PIPs to perform the updates on the AMs and release the locks. For sessions where the access decision was evaluated as ”deny”, the workflow goes to the step 21. Otherwise, the CH idles until new notification is received as in the step 15 of the continuous control phase; (21) If for a given session the end of access is received from the PEP, or the access decision after the policy re-evaluation is “deny”, then the CH computes a set of attributes which where exploited only by this section and unsubscribes for notifications from the PIPs. Also, some post-updates of attributes might be required by the policy and they are enforced in the similar way to the pre-authorization phase; (22) The CH asks the SM to change the status of the session indicating the end or revocation of the session; (23) In case of revocation, the PEP receives the revoke message from the PEP and terminates the ongoing access.

V. Implementation

We developed a preliminary prototype implementing main features of our approach. We integrated the prototype with the OpenNebula (One), an open-source middleware for managing Cloud resources and providing Cloud IaaS services.

We embedded a data usage policy (usage control, PIP and PDP allocation policies) in the extension section of the OVF (Open Virtualization Format) package of the VMI which should be protected. The usage control policy was expressed in the U-XACML language [1], [6], which is the extension of the OASIS XACML language [2], that provides specific constructs for usage control. PIP and PDP allocation policies were embedded in the OVF package in the XML format.

PEP implementation. One is extensible system and we configured it to exploit our authorization support. The PEP code was embedded in several components of the One and we refer to [7] where details are given on how the continuous control is enforced by the PEP and how One functional operations are mapped to usage control actions. Our current implementation of the PEP extends [7] in several ways. First, the PEP was enhanced with the possibility of retrieving, evaluating and enforcing PDP allocation policies. Second, the PIP allocation policy and the U-XACML policy where added into the payload of the request message which is sent by the PEP to the CH (step 2 in Figure 3).

Implementation of decision process. The PDP, the CH, the SM, and the AT were realized as a state-full authorization web-service in the Axis2 framework. Our authorization service communicates with PEPs and PIPs exploiting SAML/XACML security assertions inserted into SOAP messages sent via secured channel (HTTPS). We extended the work in [7] as follows. The CH was enhanced with the possibility of accepting and enforcing PIP allocation policies and U-XACML policies with the access request. Then, the authorization service produces attribute updates as the result of the access evaluation. Therefore, there is a 2-rounds communication between the PIP and the CH: (i) to collect and lock current attribute values, (ii) to update and unlock attributes after the access evaluation. Also, the CH deals with the distributed architecture of the authorization framework and with concurrency issues of attributes management. The CH collects attributes in predefined order avoiding possible deadlocks. Finally, the CH is able to subscribe to PIPs and listen for new attribute values. Though, we used our customized implementation of the subscribe-notify mechanism instead of adhering to the WS-Notification specification.

Implementation of attributes management. The PIP was implemented as a web-service which answers to attribute queries received from CHs. It converts SAML/XACML attribute queries into SQL statements and sends them to the AM using the JDBC driver. We chose the PostgreSQL DB as a realization of the AM (and the LM). The PostgreSQL allows an explicit locking on table- and row-levels thus it is possible to implement a customized locking protocol.

We implemented a conservative 2 phase-locking protocol in which all attributes are locked in predefined order and then released arbitrary. We assumed for simplicity that a database table corresponds to an attribute. Suppose, the CH requests $attr_1$ and $attr_2$ from the PIP. The PIP translates this request into the SQL statement and initiates a new transaction:

```
BEGIN;
  LOCK TABLE attr_1 IN ACCESS EXCLUSIVE MODE;
  LOCK TABLE attr_2 IN ACCESS EXCLUSIVE MODE;
  SELECT * FROM attr_1 ...;
```

Then, the PIP gets the response from the AM and sends it back to the CH as the SAML/XACML security assertion. Once all required attributes have been collected, the decision process is performed as usual by the PDP which provides the access response and new values for attributes. The CH sends new attribute values to the PIP which forwards them to the AM, releases locks, and ends the transaction:

```
  INSERT INTO attr_1 ...;
COMMIT;
```

The PostgreSQL also allows execution of a particular function whenever an important operation is performed on database. It is possible to attach a trigger to a table (row) and when this table (row) is modified the trigger fires and executes the specified function. The function can be written in procedural languages like PL/pgSQL or in C.

We exploited triggers written in the C language for implementing a subscribe/notify mechanism for a continuous usage control. When the CH subscribes to the PIP for updates of an attribute $attr_1$, the PIP creates the trigger:

```
CREATE FUNCTION notify()
  RETURNS trigger AS notification.c
  LANGUAGE C;
CREATE TRIGGER tucon
  AFTER INSERT OR UPDATE OR DELETE ON attr_1
EXECUTE PROCEDURE notify();
```

Whenever the attribute changes the trigger fires and informs the PIP about the change. The PIP forwards this information to all subscribers, i.e. CHs. If the trigger is not needed any more, the PIP drops it.

VI. Related Work

The UCON model of Sandhu et al. [3]–[5], is a model that encompasses and extends the existing access control models. Its main novelties are dynamic rights, and that mutable attributes of subjects' and objects', thus requiring continuous enforcement of a security policy during the access.

Recently, usage control in distributed systems [10] has become a hot topic of research. Indeed, the Cloud Computing allows users to keep and exploit theirs assets remotely at a low price. Guaranteeing security and privacy of users is still very challenging.

The PrimeLife project [12] proposed a model of a downstream usage control where data objects travel with policies attached. The approach studies distribution patterns of personal data and allows the further distribution with a more restrictive policy. The policy is based on the PPL language which expresses privacy constrains on sharing personal data.

A provenance-based access control model [13] is focused on the distribution of data objects. This model assumes that the objects might be modified by parties it visits. Also, derived objects can be generated and should be controlled.

Data usage control model proposed in [11] concerns on the data flows in the distributed environment. Particularly, it researches the intra- and cross-systems distribution modes. After the data are stored somewhere, the further access decision process is done locally.

Our model extends listed approaches in the following. First, we consider a continuous control over the data object and imply the access revocation. Second, we assume a fully distributed infrastructure which requires collaboration of many parties. Finally, in these approaches security policies are enforced using local attributes, i.e., attributes that encodes information about the data objects stored in a certain domain. These attributes are not shared with other peers and authorization systems. Therefore, it is not evident how to express and enforce the policy on all (some) copies of the same data object which requires attributes of each and every copy of the data object.

Access control model in distributed environment where many PDPs collaborate on producing the access decision is studied in [16], though we also consider usage control.

VII. Conclusions

This paper proposed the approach regulating the usage of data in distributed systems, once they have been stored outside the originator's domain. The approach is based on the UCON model, and its main advantage is that the data originator is enabled to define policies on his data to regulate their usage when these data have been released. The paper also proposed a reference architecture and its initial implementation for the enforcement of data usage policies that addresses the concurrency issues due to the concurrent execution of several decision processes in the distributed environment.

However, this paper assumes that data are always stored in trusted domains, in order to guarantee that the data usage policy paired with the data is always and promptly enforced. For future work we plan to address also the scenario where the data are stored in domain that are not trusted, and the security support must also guarantee the secure storage of the data and of the policy. Another future work concerns the introduction of the risk concept in the policies to reduce the overhead introduced by the serialization of the access to attributes. Also, we would like more formally state and prove properties that our design hold regarding concurrent retrieval and update of attributes. Finally, the prototype is going to be tested and evaluated in a real test-bed with a large number of distributed components.

References

[1] 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, 2010.

[2] OASIS XACML TC. eXtensible Access Control Markup Language (XACML) Version 3.0, OASIS Standard, 2013

[3] J. Park and R. Sandhu. Towards usage control models: Beyond traditional access control. In *SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 57–64, NY, USA, 2002.

[4] 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 (TISSEC)*, 8(4):351–387, 2005.

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

[6] A. Lazouski, F. Martinelli, and P. Mori. A prototype for enforcing usage control policies based on XACML. In *Proc. of TrustBus-12*, Springer.

[7] A. Lazouski, G. Mancini, F. Martinelli, and P. Mori. Usage control in Cloud systems. In *Proc. of ICITST-12*, 202–207, 2012.

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

[9] J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. d. Bruijn, C. d. Laat, M. Holdrege, and D. Spence. AAA authorization framework, 2000.

[10] A. Pretschner, M. Hilty, and D. Basin. Distributed usage control. *Commun. ACM 49*, pages 39–44. ACM, 2006.

[11] F. Kelbert and A. Pretschner. Data usage control enforcement in distributed systems. In *proceedings of the third ACM conference on Data and application security and privacy (CODASPY '13)*, pages 71–82. ACM, New York, NY, USA, 2013.

[12] L. Bussard, G. Neven, and F.-S. Preiss. Downstream Usage Control. In *proceedings of IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY)*, pages 22–29. IEEE, 2010.

[13] J. Park, D. Nguyen, and R. Sandhu. A provenance-based access control model. In *proceedings of the 2012 Tenth Annual International Conference on Privacy, Security and Trust (PST) (PST '12)*, pages 137–144. IEEE Computer Society, 2012.

[14] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: a virtual machine-based platform for trusted computing. *SIGOPS Oper. Syst. Rev. 37, 5*, pages 193–206, 2003.

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

[16] D.W. Chadwick, L. Su, and R. Laborde. Providing secure coordinated access to grid services In *proceedings of the 4th international workshop on Middleware for grid computing (MCG '06)*. ACM, New York, 2006