A Prototype for Enforcing Usage Control Policies Based on XACML*

Aliaksandr Lazouski, Fabio Martinelli, and Paolo Mori

Istituto di Informatica e Telematica
Consiglio Nazionale delle Ricerche
G. Moruzzi 1, Pisa, Italy
{aliaksandr.lazouski,fabio.martinelli,paolo.mori}@iit.cnr.it

Abstract. The OASIS XACML standard emerged as a pure declarative language allowing to express access control. Later, it was enriched with the concept of obligations which must be carried out when the access is granted or denied. In our previous work, we presented U-XACML, an extension of XACML that allows to express *Usage Control (UCON)*. In this paper we propose an architecture for the enforcement of U-XACML, a model for retrieving mutable attributes, and a proof-of-concept implementation of the authorization framework based on web-services.

1 Introduction

The *Usage Control (UCON)* model [4,5] extends traditional *Access Control* models to address the issues of modern distributed computing systems. UCON introduces new features in the decision process, such as the mutable attributes of subjects, objects, and environment, and the continuity of policy enforcement to guarantee that the right of a subject to use the resource holds while the access is in progress. Hence, this model can be successfully adopted in case of long-standing accesses to resources, because the access right is continuously re-evaluated during the usage of the resource and the access is interrupted as soon as this right does not hold any more. In recent years UCON has drawn a significant interest from the research community on formalization and enforcement of policies [10].

As an example, the UCON model can be successfully adopted to regulate the usage of virtual resource in the Cloud scenario, such as Virtual Machines running in Infrastructure as a Service (IaaS) Clouds. Usually, these resources are long-standing instances exposed to end-users through a proper interface. For example, a Software as a Service (SaaS) Cloud provider could implement his file storage service on top of IaaS services. In this case, the access of the file storage service provider to the IaaS service could last a very long period of time, even

* This work was supported by the EU FP7 projects *Open Computing Infrastructures for Elastic Services* (CONTRAIL) FP7-ICT 257438 and *Network of Excellence on Engineering Secure Future Internet Software Services and Systems* (NESSOS) FP7-ICT 256980.

S. Fischer-Hübner, S. Katsikas, G. Quirchmayr (Eds.): TrustBus 2012, LNCS 7449, pp. 79–92, 2012.

when the access right is not valid any more. Hence, traditional Access Control models, that aim to assure that only allowed principals are granted to access a resource [1] by performing the policy evaluation at the request time only, are not sufficient in this scenario.

In a previous work [2], we defined the U-XACML language, that is an extension of XACML [3] for expressing UCON policies. In this paper, we propose an authorization framework for the enforcement of Usage Control policies written in U-XACML and we present a proof-of-concept implementation of the proposed framework based on the web-service technology, along with some performance tests that show promising results. Also, we introduce the *attribute retrieval policy* for collecting fresh values of mutable attributes. This policy is separated from the Usage Control one because attribute retrieval is usually environment-dependent whereas the Usage Control policy is application-independent and encodes high-level security goals.

The paper is structured as follows. Section 2 gives basic notes on Usage Control, a running policy example, and describes the U-XACML language. Section 3 addresses the retrieval of mutable attributes. Section 4 describes the architecture of the prototype, its implementation details and the performance tests. Section 5 summarizes related work. Section 6 concludes the paper.

2 Usage Control

Figure 1 shows on the time axis the main difference between the traditional Access Control models and the UCON model [4,5]. The subject requests to execute a long-standing action over a resource by sending the "*tryaccess*" request. Access control models authorize the execution of the access at request time, i.e., before it starts. Assuming that the authorization framework allows to execute the access, it replies with the "*permitaccess*". Traditional Access Control models do not perform any other check from this point on. The UCON model, instead, performs security checks after the access is started. The continuous policy enforcement starts when the system, which executes the access, notifies the authorization framework by sending the "*startaccess*" message. This message identifies that a new usage session has been created and the access is ongoing. Then, the authorization framework *continuously* re-evaluates the access decision. If the policy is not satisfied any more at some point of time, the authorization framework issues the "*revokeaccess*" and forces the system to terminate the access. If the policy always holds, the system notifies the authorization framework by sending the "*endaccess*" message when the action finishes its execution. Either the usage session is revoked or ended, Usage Control concludes by performing post attribute updates and post obligations specified by the policy.

Continuity of control is a specific feature of the Usage Control model intended to deal with mutable attributes of the requesting subject, of the accessed resource and of the execution environment. Attributes change values as a result of the access execution or caused by other uncontrollable factors. Continuous control implies that policies are re-evaluated each time the attributes change their values.

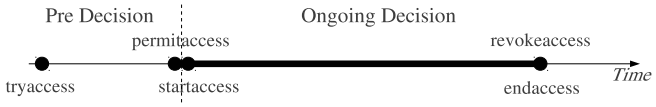


Fig. 1. Access and Usage Control

Access decisions in Usage Control are based on *authorizations* (predicates over subject and object attributes), *conditions* (predicates over environmental attributes), and *obligations* (actions that must be performed along with an ongoing access). Authorizations and conditions constrain behaviour of mutable attributes qualified in terms of time, e.g. "a subject reputation must be *always* above a given threshold during the access". The access evaluation is based only on the current values of attributes. This allows to define a mutable attribute as a *bag* in the XACML terms and model authorizations and conditions as the XACML's `<Condition>`. Obligations in Usage Control do not correspond exactly to the obligations in the XACML language. Also, attribute updates are not specified in XACML. These concepts should be added to the XACML language in order to support the full expressiveness of UCON.

2.1 Running Policy Example

Let us consider the following security policies explained in the natural language which govern operations on Virtual Machines (VM) in Cloud IaaS scenario and include both traditional Access Control and Usage Control:

- **Access Control authorizations (Usage Control pre authorizations):** A user is allowed to access to an endorsed VM, i.e., the VM certified by the producing authority;
- **Usage Control ongoing authorizations:** Users are allowed to run VMs as long as they have a high reputation and the balance of their e-wallet is positive;
- **Usage control ongoing obligations:** During the VM execution, the system sends notifications when the balance of user's e-wallet is below a threshold. These notifications repeat every 30 minutes unless the balance is recharged;
- **Usage control post updates:** If the VM execution was ended by the user, the reputation should be increased, while if the access was revoked by the authorization framework, the reputation should be decreased.

2.2 U-XACML Approach

Figure 2 shows the U-XACML policy schema that is obtained by enhancing the standard XACML language with the constructs to express when the conditions and obligations must be evaluated. To represent continuous control, U-XACML specifies when the access decision must be taken through the clause

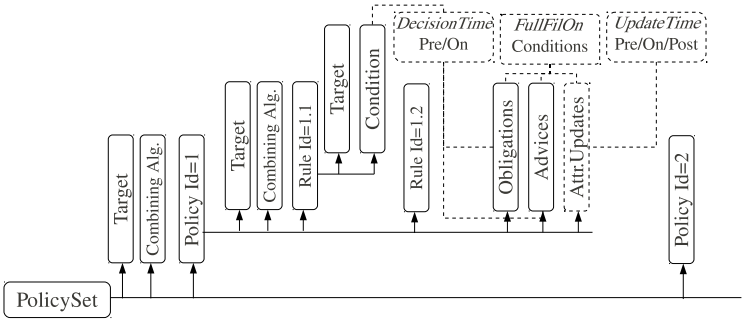


Fig. 2. U-XACML Policy Model

`DecisionTime` in the `<Condition>` elements (the admitted values are `pre` and `on` denoting, respectively, pre and ongoing decisions), and the `TriggerOn` clause in the `<Obligation>` elements. To represent attribute updates, we defined a new element, `<AttrUpdates>`, that contains a collection of single `<AttrUpdate>` elements to specify update actions. The time when the update is performed is stated by the element `UpdateTime` that has values of `pre`, `on` and `post` updates. For further details on the policy language, we refer the reader to [2].

3 Mutable Attributes Retrieval

Here and in the following, the meaning of the term "attribute" is the same as defined in UCON [4], i.e., properties paired to subjects, objects, or environment. The *role* or the *reputation* are examples of subject's attribute. The enforcement of Usage Control implies a policy re-evaluation each time when mutable attributes change their values. Catching all attribute changes is a challenging issue, usually impossible. We introduce a concept of *attribute retrieval policy* which specifies when to collect fresh attribute values and trigger the access re-evaluation.

Following the XACML approach, we propose an XML-based language to express retrieval policies. The `AttributeRetrieval` is a top-level element in the policy schema which aggregates `Target` and `Prerequisites` elements:

```
<xs:element name="AttributeRetrieval"/>
<xs:complexType name="AttributeRetrivalType"/>
  <xs:sequences><xs:element ref="Target"><xs:element ref="Prerequisites"></xs:sequences>
<xs:complexType>
```

The `Target` element specifies identifiers of mutable attributes which the retrieval policy is intended to collect. The `AttributeValue` element taken from the XACML policy schema, contains a literal value of an attribute identifier:

```
<xs:element name="Target"/>
<xs:complexType name="TargetType"/>
  <xs:sequences><xs:element ref="xacml:AttributeValue"></xs:sequences><xs:complexType>
```

The `Prerequisites` element includes a conjunctive sequence of conditions which must be satisfied before executing an attribute retrieval and the subsequent

```
<AttributeRetrieval>
 <Target><AttributeValue> u-xacml:subject:reputation </AttributeValue></Target>
 <Prerequisites>
  <Condition>
   <Apply FunctionId="dataTime-greater-then">
    <AttributeDesignator Category="environment" AttributeId="env:current-time"/>
    <Apply FunctionId="addTimeDuration">
     <AttributeDesignator Category="local" AttributeId="um:last-retrieval-time"/>
     <AttributeDesignator Category="configuration" AttributeId="time-between-queries"/>
    </Apply></Apply>
  </Condition>
 </Prerequisites>
</AttributeRetrieval>
```

Fig. 3. Attribute Retrieval Policy

re-evaluation of a usage policy. The prerequisites are done when *all* conditions are evaluated to true:

```
<xs:element name="Prerequisites"/>
<xs:complexType name="PrerequisitesType"/>
  <xs:sequences><xs:element ref="xacml:Condition"></xs:sequences><xs:complexType>
```

The `Condition` element is taken from the XACML policy schema and it expresses a boolean function evaluating the environmental factors, configuration settings and local variables.

Figure 3 shows an example of the retrieval policy which states that subject's reputation must be refreshed when x minutes passed since the last attribute retrieval, where x is represented by the configuration setting `time-between-queries`. The new attribute retrieval is performed when the current time is greater than the sum of the `time-between-queries` with the time of the last retrieval, that is stored in the local variable `um:last-retrieval-time`. This variable is updated every time when a new value of the attribute is collected.

The attribute retrieval policy is enforced when the access is in progress. When the conditions in the policy hold, the PIP is invoked to collect fresh attributes, that are then pushed to the PDP for the access re-evaluation.

4 Prototype

This section presents the architecture and the implementation details of our prototype of authorization framework supporting the enforcement of U-XACML policies.

4.1 Architecture and Work-Flow Model

The authorization framework works by intercepting every access request (e.g., Virtual Machines creation, suspension, reactivation and disposal in the IaaS Cloud scenario) determining whether the request is allowed in accordance with security policies, and enforcing the access decision by executing or aborting the request. While the access is in progress (e.g., a Virtual Machine is running) the

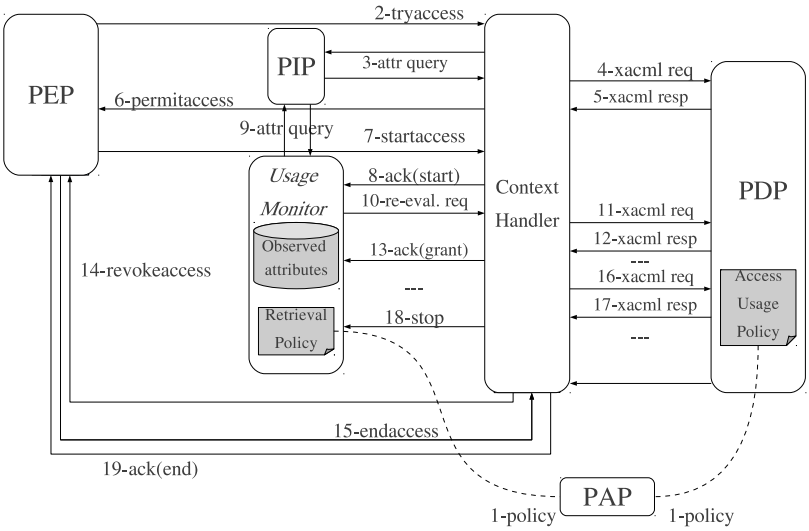


Fig. 4. U-XACML Policy Enforcement Architecture

authorization framework should be able to terminate the access and release the resource when the security policy is violated. Figure 4 shows the main components of the authorization framework's architecture. As most authorization systems [11,3], the main components are:

- *Policy Decision Point* (PDP) evaluates security policies based on the attribute values that have been included in the request;
- *Policy Enforcement Points* (PEPs) are embedded in the components of the framework that implement accesses to resources. They intercept access requests and grants or denies the access based on the decisions provided by the PDP. Moreover, PEPs are also in charge of interrupting access that are in progress to enforce a revocation decision provided by the PDP;
- *Policy Information Point* (PIP) manages *real attribute values*, and provides facilities for its storing, updating, retrieving, and delivery to the PDP;
- *Policy Administrative Point* (PAP) provides and manages security policies;
- *Context Handler* (CH) converts messages sent among components in the proper format. It is also the front-end of the authorization system, since it mediates the message exchanges with the PEPs;
- *Usage Monitor* (UM) is the main novelty with respect to the XACML architecture. This component implements the ongoing decision process by collecting fresh attribute values, and triggering the policy re-evaluation. The strategy that specifies when the new attribute values have to be collected is described by the attribute retrieval policy. The Usage Control policy re-evaluation is triggered only if the values of the attributes that have been collected are different from the ones stored in the UM cache.

The enforcement of access control is the same as described by XACML and without loss of generality we present here the simplest way to enforce it. Our authorization framework operates enforcing Usage Control policies as follows:

1. The PAP is exploited to create policies for Usage Control and attributes retrieval and makes them available for the PDP and the UM respectively;
2. The PEP intercepts the access request R_A and sends the "tryaccess" message to the CH;
3. The CH pulls from the PIP the attributes concerning the subject S_A and the resource O_A involved in R_A;
4. The CH constructs the XACML request exploiting R_A and the attributes retrieved in the previous step, and sends it to the PDP;
5. The PDP evaluates the security policy rules concerning traditional access control rules, and returns the XACML response to the CH;
6. The CH translates the PDP response and replies to the PEP. Let us assume that the response is "permitaccess". Hence, the PEP starts the access A;
7. The PEP notifies the CH that a new usage session was created and the access has started;
8. The CH activates a new instance of UM for A, and it forwards the request to this UM instance that, in turn, starts the continuous policy enforcement by monitoring the attributes related to S_A and O_A;
9. The UM enforces the retrieval policies related to the attributes of S_A and O_A to decide when fresh values must be pulled from the PIP;
10. If the observed values are different form the cached ones, the UM triggers the policy re-evaluation sending the request to the CH (go to the next step). Instead, if no changes in the attribute values are detected, the UM repeats the previous step by enforcing the attribute retrieval policy again and deciding when fresh attributes values must be pulled from the PIP;
11. The CH constructs the XACML request to re-evaluate the right of executing A and sends it to the PDP;
12. The PDP evaluates the security policy rules concerning ongoing control rules, and returns the XACML response to the CH;
13. If the response is "deny", see the step 14. Otherwise, "grant" is received and the access can be continued, thus the CH translates it and replies to the UM. Steps 9-13 are repeated until either the PDP returns "deny" or the PEP sends the "endaccess" message: in this case go to the step 15;
14. If the PDP returns "deny", the CH sends the access revocation message to the PEP. Then, the PEP terminates A;
15. The PEP also may stop the session due to the normal ending of access. The PEP notifies the CH by sending the "endaccess" message;
16. The CH constructs the XACML request to the PDP;
17. The PDP evaluates the security policy rules concerning end of access, and returns the XACML response to the CH. The response could specify some post obligations or post updates that must be executed;
18. The CH destroys the running UM instance which monitors A;
19. Finally, the CH responses to the PEP. This message includes the request for executing post obligations, if any.

```
<PolicySet PolicyCombiningAlgId="deny-override" ... >
 <!--  - Access control Policy --!><Policy ... ><Rule Effect="Permit" ... >
     <Target><AttributeValue> TRYACCESS </AttributeValue> ... AttributeId="action-id" ... </Policy>

 <!-- Usage control: authorizations and revocation --!>
 <PolicySet PolicyCombiningAlgId="permit-override" ... >
   <Target><AttributeValue> STARTACCESS </AttributeValue> ... AttributeId="action-id" ...
    <!-- permit part --!><Policy RuleCombiningAlgId="permit-override"... >
      <Rule Effect="Permit" ... <Condition> ... </Condition></Rule>
      <ObligationExpressions ... ObligationId="call-me-back" FulfillOn="permit" ... </Policy>

    <!-- deny part --!><Policy RuleCombiningAlgId="deny-override"... >
      <Rule Effect="Deny" ... > ... </Rule>
      <ObligationExpressions ... ObligationId="post-update" FulfillOn="deny" ... </Policy>
 </PolicySet>

 <!-- Usage control: ongoing obligations and updates --!>
 <Policy RuleCombiningAlgId="permit-override"... >
   <Target><AttributeValue> STARTACCESS </AttributeValue> ... AttributeId="action-id" ...
   <Rule Effect="Permit" ... > ... </Rule>
   <ObligationExpressions ... ObligationId="ongoing-update" FulfillOn="permit" ... </Policy>

 <!-- Usage control: end of access --!><Policy RuleCombiningAlgId="permit-override"... >
   <Target><AttributeValue> ENDACCESS </AttributeValue> ... AttributeId="action-id" ...
   <Rule Effect="Permit" ... ></Rule>
   <ObligationExpressions ... ObligationId="post-update" FulfillOn="permit" ... </Policy>
</PolicSet>
```

Fig. 5. Access and Usage Control Policy in XACML Syntax

4.2 Implementing U-XACML Policies Exploiting XACML

The implementation of the framework for the enforcement of the U-XACML policies exploits existing engines for the evaluation of XACML policies. Since U-XACML is based on XACML, we implemented the continuous control combining original XACML constructs. The basic idea is to insert a looping construct of the access re-evaluation inside XACML's obligations. An obligation is an action which must be executed by the PEP along with the enforcement of an access decision. The semantics of obligations is not fixed by XACML, thus it can be exploited to encode any kind of duties. Hence, we exploit obligations for implementing Usage Control. In particular, the PDP evaluates the policy and, in case of an ongoing condition, it sends the access decision along with a so-called "call-me-back" obligation which forces the PEP to call the PDP again to trigger the access right re-evaluation. This routine is repeated while the policy is satisfied and the access does not terminate. When the policy is violated, instead, no "call-me-back" obligation is returned.

We represent a U-XACML policy exploiting a tuple of 5 XACML policies. One of these policies encodes the traditional access control (i.e., pre decisions), while the other policies express Usage Control features and represent: (U-OAC) ongoing authorization and conditions, which must be satisfied during the usage; (U-OBU) ongoing obligations and attribute updates; (U-PBUR) post-obligations and updates in a case of access revocation; (U-PBUE) post-obligation and updates in a case of a normal termination. Figure 5 contains an overall UCON policy structure in XACML syntax. Notice, that non-critical elements are

omitted due to space limitation. This structure should be preserved and a policy should abide the following requirements:

- Access control policy should include all predicates over static attributes since there is no need to re-evaluate these predicates when the access is ongoing;
- Usage policy for ongoing authorizations and conditions should be modeled as a policy set with two sub-policies: permitting and denying. The permitting policy, i.e., the U-OAC policy, should contain a "call-me-back" obligation, whereas the denying policy, i.e., the U-PBUR policy, includes post-updates and post-obligations that must be executed in a case of a policy revocation. Ongoing authorizations may be placed in any sub-policy. If a policy designer is more comfortable to specify conditions which must always hold during the access then these conditions are included in the permitting policy, and if it is more intuitive to express negative rules - to the denying policy;
- Usage Control policy for end of access, i.e., the U-PBUE policy, should contain no conditions in a policy rule and the rule's effect should be "permit". The policy applicability is defined by the action attribute "action-id" with the value equals to "endaccess";
- Obligations returned to the PEP should be a composition of obligations whose "FulfillOn" attribute and the effect of a policy evaluation matches with the effect returned after the evaluation of the overall policy.

We transformed policies written in U-XACML to XACML using the XSLT technology[1]. In fact, this transformation adheres to the requirements listed here and the final policy ported for the evaluation looks like one given in Figure 5. Since we implemented U-XACML policies exploiting standard XACML constructs, one might question whether U-XACML is really needed. The main reason is that the U-XACML policy allows to naturally express Usage Control features with specific constructs, hence it is more user-friendly. System administrators can transform their access control policies written in XACML in Usage Control policies with a minimal effort. They should identify which of the conditions and obligations in their access control policies must hold during the usage of the resource, and insert the clause `DecisionTime="on"` in the XACML element that declares these conditions and obligations.

Moreover, the proposed use of the original XACML constructs for implementing Usage Control has some drawbacks. Firstly, the PEP, besides being responsible for the enforcement of the PDP decisions, is also in charge of iteratively triggering the PDP for the policy re-evaluation. Secondly, a "call-me-back" obligation may contain a policy-sensitive data and its disclosure is unwanted. Therefore, we introduced the UM, a new component, which enforces the call-me-back obligation by retrieving fresh attributes. The prototype given in this paper has a simple implementation of Usage Control which is based on standard techniques and shows a relatively good performance.

[1] `http://xml.apache.org/xalan-j`

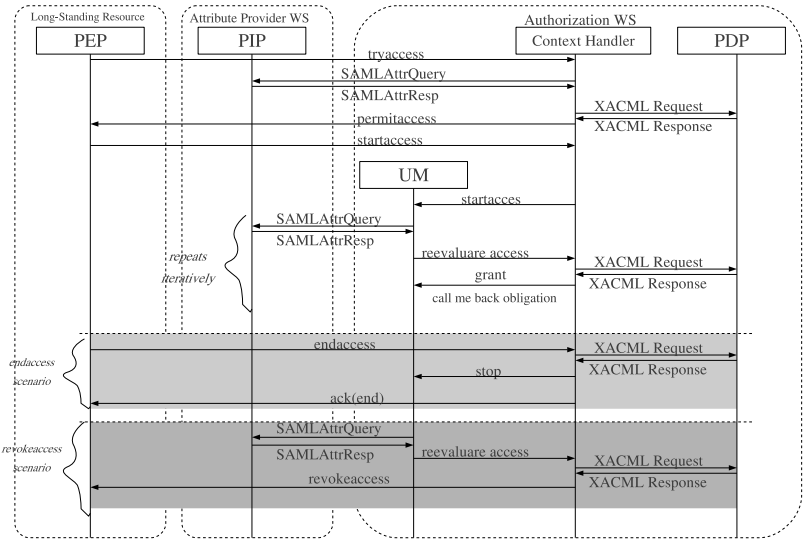


Fig. 6. A Sequence Diagram of Access and Usage Control Enforcement

4.3 Prototype Implementation

The core of our framework is the *authorization service* which implements functionality of the UM, the CH, the PDP, the PAP, and it was realized exploiting the web service technology (see Figure 6). An instance of the authorization service is created per an instance of the usage session. The service instance is destroyed when the violation of the policy happens or the usage session ends normally. This is a preliminary implementation of the framework, where all concurrent usage sessions run in parallel and their monitors do not cooperate on attributes retrieval.

We exploited the Sun's XACML engine[2] for implementing the PDP. This engine evaluates access and usage policies. At this stage, we do not provide an engine for the evaluation of the attribute retrieval policy. We leave it for the future work. Instead, in the current implementation we provide the possibility to retrieve all attributes repeatedly, waiting a time interval (e.g., every 10 seconds).

The PIP manages real attributes and provides interfaces to query their values. The PIP was realized as a web service and its clients are the authorization service and the PEP. The PIP communicates over HTTP/SOAP and accepts the *SAML attribute queries* as requests and replies with the *SAML assertions*. We used the OpenSAML2.0 Extension Library[3] to support the SAML profile of XACML[4].

The PEP is a process which runs along with the execution of the long-standing action on the resource. The PEP enhanced with Usage Control features should be

[2] `http://sunxacml.sourceforge.net`
[3] `http://www.bccs.uib.no/~hakont/SAMLXACMLExtension`
[4] `http://saml.xml.org/saml-specifications`

powerful to destroy the running action if the authorization service claims about a policy violation or to notify the authorization service if the execution of the action ends normally. We provided a set of Java APIs which the designer of a system should use for the Usage Control support. These APIs implement the communication between the PEP and the CH, the PEP and the PIP with the support of the HTTP/SOAP/SAML-XACML protocol stack. The PEP is the Axis2[5] client of the authorization and attribute provider services. Since the time of the usage session in unbounded, the PEP communicates with the authorization service in *asynchronous* mode which enables the processing of the "startaccess" and "revokeaccess" messages in two different threads. Also, two different transport channels are used to send these messages. The PEP starts the usage session after sending the "startaccess" and idles. Later, when the response from the authorization service about the policy violation arrives, the PEP resumes its execution by destroying the usage session.

Figure 6 presents a sequence diagram for the enforcement of the access and usage policy given in Subsection 2.1. It starts with the access control and the CH is responsible to collect attributes. When the usage sessions begins, the UM becomes in charge for the retrieval of fresh attribute values. When new attribute values violate a security policy, the authorization service informs the PEP and both terminate. In case of normal end of access, the new instance of the authorization service is created to process the "endaccess" request. Before replying to the PEP, this instance stops the primary running instance of the authorization service created on the "startaccess".

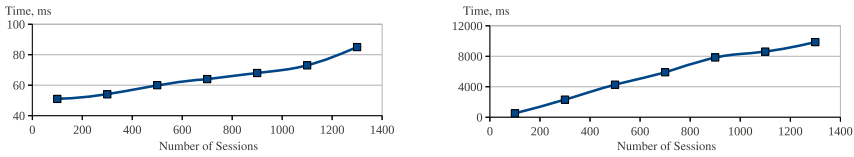
4.4 Performance Evaluation

As a Usage Control requires an extra process for each usage session, the performance should be considered. Since this process is outsourced to the authorization service, we measured its performance in the presence of a plenty of running usage sessions. We deployed the authorization and attribute provider services inside Axis2. The server was hosted on a machine with Ubuntu 10 and Java 1.6 support and which has Intel Core 2 Duo 3.16 GHz CPU and 3.4 GB memory.

First, we measured how many instances of the authorization service per second can be created. This gives a number of usage sessions which the PEP can start and be aware that the authorization service will serve them all. The creation of the authorization service instance starts when the PEP sends "startaccess" and lasts until the UM receives the result of the first access reevaluation (i.e., steps 7-13 in Figure 4). We obtained that 47 new usage sessions in average can be created by the authorization service, or approximately 21.5 ms goes for the creation of a single service instance. Although, we experimented with a faster CPU the obtained results shown that our system performs comparably to one given in [12] where the average time per access evaluation only takes 45.3 ms.

Then, we measured the *revocation response time* for a single usage session in dependence on the number of ongoing usage sessions serviced by the

[5] `http://axis.apache.org/axis2`



(a) Revocation of a Single Session (b) Massive Revocation of Sessions

Fig. 7. Performance of Access Revocation

authorization service. The revocation response time defines the period of time passed from the point when the PIP replies to the UM with the attribute value which violates the security policy until the PEP receives the "revokeaccess" (i.e., steps 9-14 in Figure 4). We varied the number of the authorization service instances running concurrently with the test usage session from 100 to 1300. We assume that an execution time of a single session is quite long, thus the authorization service could maintain a large number of concurrent sessions started at different time points. Since the authorization service can create only 47 new sessions per second, the starting time between 1st and 1300th sessions and the minimum execution time of any session should be at least 26 seconds. Figure 7(a) shows the results obtained. We see that the response time of the access revocation is moderate and increases slowly with the growth of the number of concurrently running sessions.

Finally, we measured the revocation time of all sessions whose policies use the same security attribute which changes its value from good to bad and violates the policies. The revocation time of all sessions defines the period of time passed from the point when the first until the last sessions receive the "revokeaccess". We configured the retrieval policy in such a way that the UM refreshes attributes and re-evaluates the usage policy every *10* seconds. Figure 7(b) shows the results obtained. We see that the revocation time of all sessions grows linearly in the number of running sessions.

5 Related Work

Several papers stated that XACML needs extension to capture the continuous policy enforcement [12,8]. Some attempts were done to enforce UCON policies exploiting XACML [9,12,6]. These approaches introduce events reporters that trigger the policy re-evaluation when the access is in progress. Security checks are invoked by the changes of subject, object, and/or environmental attributes. Instead, we assume that the authorization framework is responsible to retrieve fresh attributes. Moreover, they consider what parts of UCON can be modelled in XACML. In contrast, our approach considers how XACML should be extended to capture the continuous control and we introduced the prototype which is capable to deal with the main UCON features. The approach given in [7] proposed to integrate together the attribute retrieval and security policies

in a single XACML policy. We, instead, separate them because the attribute retrieval is usually environment-specific while security policies are not and usually encode high-level security goals.

6 Conclusions and Future Work

This paper presented an authorization framework for the enforcement of Usage Control policies expressed with the U-XACML language, along with a web-services based proof-of-concept implementation that, although very simple, showed promising results from the performance point of view. The main advantage of the proposed framework is that it supports U-XACML, thus simplifying significantly the enforcement of Usage Control. As a matter of fact, exploiting U-XACML, system administrators can transform their access control policies written in XACML in Usage Control policies in a straightforward way. In particular, they should identify which of the conditions and obligations in their access control policies must hold during the usage of the resource, and insert the clause `DecisionTime="on"` in the XACML element that declares these conditions and obligations.

Another advantage of the proposed framework is that most of the interactions between the framework components are implemented through standard protocols, thus allowing the substitution of the existing component with enhanced (e.g., more efficient) ones.

We are currently working for refining several aspects of the authorization framework. Firstly, we are working on refining the support for attributes retrieval policies, e.g., we are implementing publish/subscribe retrieval policies. Moreover, we would like move to a *state-full* version of the authorization service implementing an architecture with a single Usage Monitor component that will manage all the ongoing accesses. We believe that these changes will enhance the framework performance and robustness especially in the case of a large number of long lasting concurrent accesses.

References

1. Abadi, M.: Logic in access control. In: Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science, Washington, DC, USA, p. 228 (2003)
2. Colombo, M., Lazouski, A., Martinelli, F., Mori, P.: A proposal on enhancing XACML with continuous Usage Control features. In: Proceedings of CoreGRID ERCIM Working Group Workshop on Grids, P2P and Services Computing, pp. 133–146. Springer (2010)
3. OASIS XACML TC. eXtensible Access Control Markup Language (XACML) Version 3.0 (2010)
4. Park, J., Sandhu, R.: Towards usage control models: Beyond traditional access control. In: SACMAT 2002: Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies, NY, USA, pp. 57–64 (2002)
5. Zhang, X., Parisi-Presicce, F., Sandhu, R., Park, J.: Formal model and policy specification of usage control. ACM Transactions on Information and System Security (TISSEC) 8(4), 351–387 (2005)

6. Feng, J., Wasson, G., Humphrey, M.: Resource usage policy expression and enforcement in grid computing. In: IEEE/ACM International Workshop on Grid Computing, pp. 66–73 (2007)
7. Gheorghe, G., Crispo, B., Carbone, R., Desmet, L., Joosen, W.: Deploy, Adjust and Readjust: Supporting Dynamic Reconfiguration of Policy Enforcement. In: Kon, F., Kermarrec, A.-M. (eds.) Middleware 2011. LNCS, vol. 7049, pp. 350–369. Springer, Heidelberg (2011)
8. Hafner, M., Memon, M., Alam, M.: Modeling and enforcing advanced access control policies in healthcare systems with Sectet, pp. 132–144 (2008)
9. Katt, B., Zhang, X., Breu, R., Hafner, M., Seifert, J.-P.: A general obligation model and continuity: enhanced policy enforcement engine for usage control. In: SACMAT 2008: Proceedings of the 13th ACM Symposium on Access Control Models and Technologies, New York, USA, pp. 123–132 (2008)
10. Lazouski, A., Martinelli, F., Mori, P.: Usage control in computer security: A survey. Computer Science Review 4(2), 81–99 (2010)
11. Vollbrecht, J., Calhoun, P., Farrell, S., Gommans, L., Gross, G., de Bruijn, B., de Laat, C., Holdrege, M., Spence, D.: AAA authorization framework (2000)
12. Zhang, X., Nakae, M., Covington, M.J., Sandhu, R.: Toward a usage-based security framework for collaborative computing systems. ACM Transactions on Information and System Security (TISSEC) 11(1), 1–36 (2008)