A Proposal on Enhancing XACML with Continuous Usage Control Features*

Maurizio Colombo, Aliaksandr Lazouski, Fabio Martinelli, and Paolo Mori

Abstract Usage control (UCON) proposed by R. Sandhu et al. [8, 9] is an attribute-based authorization model and its main novelties are mutability of attributes and continuity of control.

OASIS eXtensible Access Control Markup Language (XACML) [10] is a widely-used language to write authorization policies to protect resources in a distributed computing environment (e.g. Grid). The XACML policy specifies before-usage authorization process optionally complemented with obligation actions fulfillment. By now, XACML has insufficient facilities to express continuous usage control afterwards an access was granted and started.

In this paper, we introduce U-XACML, a new policy language, which enhances the original XACML with the UCON novelties. We extend a syntax and semantics of the XACML policy to define mutability of attributes and continuity of control. We introduce an architecture to enforce the U-XACML policy.

Key words: Access control, usage control, policy language, XACML, UCON, Grid computing

Maurizio Colombo, Fabio Martinelli, Paolo Mori
Istituto di Informatica e Telematica, Consiglio Nazionale delle Ricerche, via G. Moruzzi 1, Pisa, Italy e-mail: \{maurizio.colombo,fabio.martinelli,paolo.mori\}@iit.cnr.it

Aliaksandr Lazouski
Universita di Pisa, via B. Pontecorvo 3, Pisa, Italy e-mail: lazouski@di.unipi.it

* This work has been partially supported by the EU project FP6-033817 GRIDTRUST (*Trust and Security for Next Generation Grids*).

1 Introduction

Over the last decade researchers have shown advantages of attribute-based access control. Access decision is based on attributes of a requesting subject, of an accessing object, and of an environment where the computing system operates. Traditional authorization scenarios assume that attributes remain invariable in time. Therefore, authorization conditions can be checked only once before granting an access and they will also hold when the access is in progress. To express comprehensive usage scenarios R. Sandhu et al. [8, 9] proposed the UCON model and an idea of the attributes mutability occurred as a side-effect of the object usage. This idea goes beyond a before-usage authorization and assumes a continuous evaluation of a security policy before granting an access, and when the access is in progress. Essentially, this is important for long-lived accesses, which are peculiarities of Grid computing, e.g. computational services [6, 7, 11]. During last years, UCON has gathered a lot of attention as the most expressive attribute-based access control model.

XACML [10] is an OASIS standard for expressing, combining and managing access control policies in a distributed environment. XACML was designed for attribute-based access control and it is widely used nowadays in many applications and environments due to its extensibility and interoperability. By now, XACML has facilities to express traditional access control.

Several papers stated that XACML needs extension to capture the concept of access decision continuity [4, 11]. Recently, some attempts were done to implement a continuous policy enforcement using XACML [3, 5, 11]. These approaches introduce events that trigger the policy reevaluation when the access is in progress. Security checks are invoked by changes of subject, object, and/or environmental attributes. The focus of these papers was on an architecture and policy enforcement mechanisms, while a few attention was placed on the policy model expressing the UCON model. Moreover, they consider how the XACML is capable to model the UCON. In contrast, our approach considers how XACML should be extended to capture the UCON model.

The aim of this work is to present a flexible policy language called U-XACML, which enhances the original XACML with the UCON novelties. The policy language is designed towards a service oriented architecture and collaborative computing, e.g. the Grid. We analyzed semantics of the XACML policy and the UCON abstract model. We revised the notion of authorizations, obligations, conditions and attribute update actions to establish a unified semantics for the U-XACML. The U-XACML captures attribute updates, continuous policy evaluation, and places conditions triggering ongoing attribute updates and obligations. The original XACML architecture and authorization dataflow was modified to enforce the U-XACML policy. We studied several architectural approaches and shown their advantages and disadvantages.

This paper is organized as follows. Section 2 gives background on XACML and UCON. In section 3 we introduce syntax and semantics of the U-XACML policy language. Section 4 shows an architecture to enforce a continuous usage control.

Section 5 introduces a usage scenario and informal security policy for Grid computational service. We summarize the paper in Section 6.

2 Background

2.1 UCON

UCON [5, 8, 9] is a novel access control model that addresses problems of modern distributed environments. UCON has two novel aspects: 1) mutability of attributes, and 2) continuity of policy enforcement. Mutability of attributes means that attributes may change in time. Since attributes are used for access decision evaluation, policy statements should be reevaluated continuously whenever any change of any attribute value occurs. UCON implies security policy enforcement not only when the request for an object should be authorized, but also when the actual usage of the object is in progress. This continuous evaluation is particularly important in the case of long-lived accesses, i.e. accesses that lasts hours or even days (e.g. Grid services). If during the usage attribute change violates a security policy, the access has to be revoked and the resource's usage terminated.

The UCON policy is stated using authorizations, obligations and conditions. Authorization predicates put constrains on subject's and/or object's attributes (e.g. subject's name must be "John"). Obligations are mandatory actions performed by an obligation subject (not always a requesting subject). Conditions are environment restrictions (e.g. object is available during working hours).

Generally, the UCON policy is paired with the following meta information:

- *Access decision factor*. A security policy can be stated using authorizations only, obligations only, conditions only, or any of their combinations. These policy statements are encoded by A, B, and C respectively.
- *Access decision timing* specifies when an access decision is evaluated. If the access decision is made before the usage, these scenarios are encoded with a prefix "pre". Access decisions evaluated during the usage are encoded with "on".
- *Attribute update timing* identifies when attributes are updated by the authorization system. Attributes can be updated before the usage (the pre-update action is encoded by (1)), during the usage (on-update action is encoded by (2)), or when the usage is over (post-update action is encoded by (3)). If no attribute updates are needed, the corresponding models are encoded by (0).

Based on the policy meta-information, the UCON model launches 24 core scenarios - combining "pre" and "on" evaluation of authorizations, obligations and conditions with attribute updates that can be performed before (1), during (2), after (3) the access, or no updates (0). For example, the "preA3" model, called as preauthorization with post-updates, represents a scenario where an access decision is evaluated only once before the usage starts. The access decision is done by check-

ing authorization predicates only, and one or more attributes are updated after the resource usage ends.

2.2 XACML

Although, the UCON proposes a comprehensive access control model, it still remains abstract. In contrast, XACML [10] is a widely-used standard to write and manage security policies. Arbitrary attributes can be expressed which makes XACML application-independent and extensible to accommodate the specific requirements of a specific application and domain. This also encourages an interoperability between components of the authorization system. Currently, the OASIS is working on XACML v3.0 draft, while several implementations of XACML v2.0 have been presented recently by third-party vendors[2].

XACML is facilitated to express the classical access control model, where the access decision is evaluated only once when a request to access a resource comes. The XACML model implies a security policy managed by multiply parties. XACML provides algorithms to build a resulting policy and assumes that many points of enforcement exists in distributed system. Access decision is based on attributes which characterize a subject, object, environment and also the content of the accessing object. A set of mathematical operators can be used over attributes to build authorization predicates. Also, the XACML model expressiveness is enhanced by introducing obligations, a set of actions performed in conjunction with the policy enforcement. XACML v3.0 is enhanced with advices, a supplementary information (not mandatory for execution) for the policy enforcement point (PEP).

XACML standards a policy meta-model, syntax and semantics, and enforcement architecture. The top-level policy elements are `<PolicySet>`, `<Policy>`, and `<Rule>`. The `<Rule>` has three main parts `<Target>` which denotes rule's applicability to the authorization request, `<Condition>`s which are authorization predicates over attributes, and effect is the result of the rule evaluation. It returns either "Permit" or "Deny" if the rule is satisfied and "Non Applicable" if the `<Target>` and/or `<Condition>`s are not satisfied. The XACML `<Policy>` consists of one or more `<Rule>`s and forms an the authorization decision accompanied with a set of `<Obligations>`. The `<Policy>` has a procedure for combining the result of the evaluation of `<Rule>`s it contains. XACML identifies several combining algorithms: deny-overrides, permit-overrides, first-applicable and only-one-applicable. The `<PolicySet>` is an optional element which provides the resulting policy through the combining of several `<Policy>`s applicable to the access request.

XACML realizes the enforcement architecture and abstract dataflow model compliant with IETF and ISO standards. Access request goes through an abstract component PEP. The PEP invokes a policy decision point (PDP) for an access decision. The PDP works as a double "filtering" of the initial access request. First, the PDP

[2] http://sunxacml.sourceforge.net, http://mvpos.sourceforge.net

evaluates an applicability of the policy to a given request (<Target> element is satisfied). In the second phase, a set of <Condition>s is checked. The resulting authorization decision plus set of <Obligations> and <Advice>s are sent back to the PEP. After fulfilling <Obligations>, the PEP grants or denies access to the resource and no further control is provided when the access is in progress. The interactions between the PDP and the PEP are synchronous and state-less.

3 U-XACML policy syntax and semantics

In this section, we propose syntax and semantics for U-XACML, an enhancement of the XACML standard that includes facilities to express UCON policies. The U-XACML policy meta-model derived from the original XACML and enhanced with UCON novelties is presented in the Figure 1. The original model was extended to capture attribute updates, continuous policy evaluation, and to place conditions triggering ongoing attribute updates and obligations.

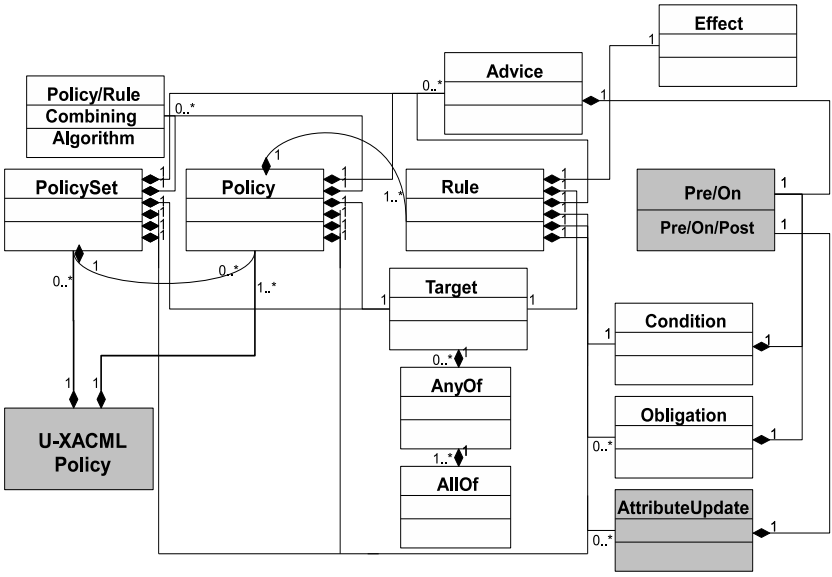

Fig. 1 U-XACML policy language model.

We start from an authorization request. Subject, object and right of the UCON model are represented in the XACML by subject, resource and action respectively. The <Target> element specifies a policy applicability to the authorization request.

Subjects, resources and environment are associated with a set of attributes. Attributes are presented in both models and the XACML identified the following elements to describe attributes: <AttributeDesignator> (states to whom and by whom the attribute is issued), <AttributeSelector> (states where the attribute can be found), and <AttributeValue> (contains an attribute value). In the XACML attributes have static values and are never changed by the policy. In contrary, the UCON model presents an attribute mutability occurred as a side-effect of the policy enforcement. Also, attributes can be changed by the execution environment. We categorizes the U-XACML attributes as follows:

- (i) Immutable, e.g. identity. These attributes are static during the usage and can be change only by the administrative actions.
- (ii) Mutable by the security policy (enforceable mutability). These attributes are changed as the result of the policy enforcement only. As example, the subject attribute could be the number of applications currently running on behalf of the subject in Grid.
- (iii) Mutable by the execution environment (observable mutability). These attributes are mutable by their nature (e.g. environmental attributes), but how they change is not stated explicitly in the policy. As example, consider time-based attributes, e.g. resource usage time, current time value, or location-based attributes, etc.
- (iv) Combination of the last two (observable mutability). These attributes can be change either by the policy, or by the environment (e.g. a reputation). Here, we also consider attributes mutable by the security policy, but the update of this attribute is triggered by the environmental attribute (e.g. in the case study presented in section 5, the time quota is updated by the policy, but when the update occurs depends on the application execution time).

This attribute mutability category is optional and can be useful during the policy enforcement phase (we refer the reader to Section 4).

To represent attribute updates, we defined a new element, <AttrUpdates>, that contains a collection of single <AttrUpdate> elements to specify update actions in the U-XACML:

```
<xs:element name="AttrUpdates" type="u-xacml:AttrUpdatesType"/>
<xs:complexType name="AttrUpdatesType">
    <xs:sequence>
        <xs:element ref="u-xacml:AttrUpdate"
            maxOccurs="unbounded"/>
    </xs:sequence>
    ...
</xs:complexType>
```

<AttrUpdate> element defines a sequence of update functions which can run in UpdateTime. In general, each of these functions refers to a distinct attribute, and defines: the name of the attribute to be updated, the mutability category of the attribute, and the update function to compute the new value of the attribute.

UpdateTime has values 1, 2, or 3 that denote, respectively, pre-, on- and post-updates:

```
<xs:element name="AttrUpdate" type="u-xacml:AttrUpdateType"/>
<xs:complexType name="AttrUpdateType">
    <xs:sequence>
        <xs:element ref="u-xacml:UpdateExpression"
          minOccurs="0"/>
    </xs:sequence>
    ...
    <xs:attribute name="UpdateTime" type="xs:integer"
      use="required"/>
    <xs:element ref="TriggerOn"/>
    ...
</xs:complexType>
```

In the previous definition, <UpdateExpression> is a specific update function.

The U-XACML policy inherits the policy structure from the XACML. The U-XACML policy contains a set of <Rule>s and each <Rule> besides update actions includes a set of decision factors. Authorizations and conditions proposed in the UCON are modeled in the XACML by means of the <Target> and <Condition>s elements. In the U-XACML, we assume that the <Target> element puts constraints on the immutable attributes only, while the <Condition>s element covers other attributes of the subject, object and environment.

Obligations. In the UCON model, obligations are actions that have to be performed by obligation subjects over obligation objects. In XACML, instead, obligations have different semantics, and are considered as duties that will be performed by the PEP enforcing the access decision.

XACML obligations can easily be exploited to implement UCON obligations when the obligation subject and object are under the control of the protection system, and the PEP can assure the obligation fulfilment. In U-XACML, we also extended XACML obligations to model UCON obligations whose subject or object reside outside the protection system. However, in this case, the PEP can only observe the obligation fulfilment by checking some conditions. Such UCON obligations are modelled as two U-XACML obligations. The first asks the obligation subject to fulfil the obligation, and the second one checks the fulfilment. This check is triggered when some conditions hold (e.g. deadline to fulfil the obligation is over or some attributes changed). XACML obligations take a set of attributes as input parameters. Due to XACML extensibility, these parameters can state a UCON obligation subject and object.

Continuous policy enforcement. In the UCON access decision should be evaluated not only before granting the access but also continuously when the access is in progress. The U-XACML policy specifies when the access decision is made by the DecisionTime in the <Obligation> and <Condition> elements. For example, the <Condition> element in the U-XACML should include the following:

```
<xs:element name="Condition"/>
<xs:complexType name="ConditionType">
    ...
    <xs:attribute name="DecisionTime" type="xs:string"
       use="required"/>
```

```
    ...
</xs:complexType>
```

DecisionTime has values pre and on. For the pre models, the conditions are evaluated only once, while for the on-going models they have to be valid permanently. Meanwhile, the obligations are duration-less in both scenarios. For the on model, obligations invocation is triggered by some conditions (e.g. in the case study presented in section 5, if application's execution time is 1 hour to reach the quota value, the ongoing obligation is triggered). The same observation applies for ongoing attribute updates, they are duration-less and triggered when some conditions hold. <ObligationExpression> and <AttrUpdate> elements contains a <TriggerOn> element:

```
<xs:element name="ObligationExpression"/>
<xs:complexType name="ObligationExpressionType">
    ...
    <xs:element ref="TriggerOn"/>
    ...
</xs:complexType>
```

The element <TriggerOn> states conditions which trigger the update and obligation actions. As a matter of fact, the <TriggerOn> has the same syntax as the <Condition>.

4 U-XACML architecture

This section presents an architecture to enforce the U-XACML policy and a data-flow between its main components (see Figure 2). The U-XACML extends the functionality of authorization components of the XACML to handle continuous policy enforcement. The policy enforcement can be separated in two parts: pre-authorization, and ongoing usage.

For pre part, we have authorization/condition predicates evaluation, attributes update and obligations fulfilment. All this actions are duration-less and executed in the order as stated above: first predicates evaluation, than updates and finally obligations. We fix this order whether the UCON assumes any possible combination. We will address this issue in the future work presenting the formal model of the U-XACML policy. The data-flow in pre-authorization is the same as in the original XACML excepting the attribute updates. The PDP invokes an Attribute Manager to update attributes before sending the access decision and obligations to the PEP.

In the ongoing model, we assume that authorization and condition predicates should be valid during all the usage, while obligation and update actions still remain duration-less. Obligations and updates are conditioned by attributes and are triggered when they change appropriately. The U-XACML ongoing policy can be represented by the sequence of states and the state is specified by attribute values. The policy state transition and decision reevaluation are driven as the result of attributes mutability. This schema is similar to the run-time monitoring of applications where the PEP intercepts security-relevant actions performed by applications and

triggers the PDP for the policy reevaluation. In usage control, instead of security-relevant actions there are security-relevant events, i.e. the change of an attribute value. The U-XACML architecture should be enhanced with a component (the Attribute Manager) which captures all security-relevant events and keeps fresh values of attri butes. Whenever an attribute is changed the PDP performs a policy check.

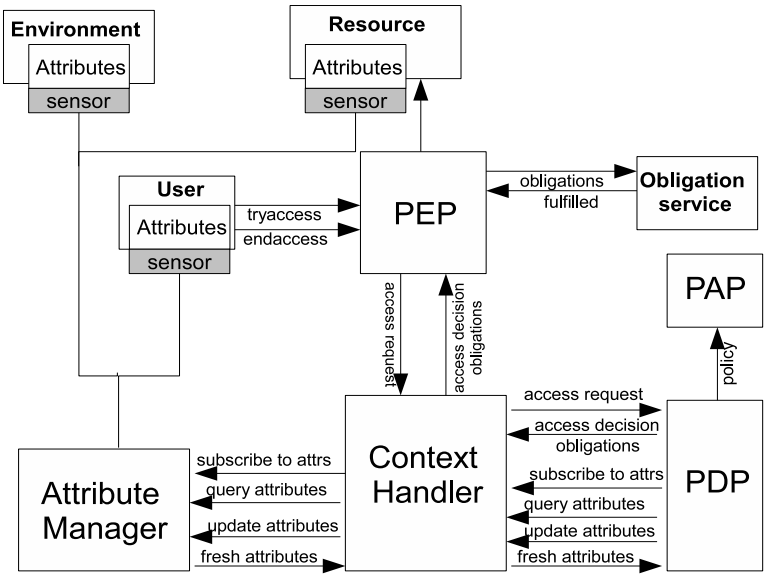

Fig. 2 U-XACML architecture and data flow.

As shown in the Figure 2 the U-XACML architecture consists of the following components (we omit the explanation of components if they functionality coincide with the original XACML):

- PEP is responsible for the enforcement of the access decision and the obligations fulfillment. The PEP can also manage meta-information associated with the usage session, e.g. session id and the current state of the policy.
- PDP receives as an input the policy, a current policy state, a set of fresh attributes. The PDP evaluates the policy associated with the current policy state. If the policy is violated the PDP return “Non applicable”. Otherwise, it returns “Permit” and makes a state transition if needed. The PDP may also invoke attribute update and/or obligation actions.
- Attribute manager has fresh values of attributes. It enforces update actions triggered by the PDP and is capable to capture any change of attributes done by the execution environment and/or the policy administration point (PAP). The Attribute Manager exploits sensors to collect attributes.

- Sensors are plugged into subject, object and environment attribute repositories. They are activated every time when attributes are changed. The new value is forwarded to the Attribute Manager.
- Context handler converts messages circulating in the system to the proper format.

Depending on how fresh attributes are acquired and security-relevant events are detected in the system, and who triggers the policy reevaluation, we assume three possible data-flow models: (i) active Attribute Manager; (ii) passive Attribute Manager with active PEP; (iii) passive Attribute Manager with active PDP.

Active Attribute Manager (push attribute model). This is an event-based model. It assumes that sensors are activated every time whenever attributes are changed. This information is forwarded to the active Attribute Manager who invokes the PDP for the policy reevaluation. It more details the ongoing data-flow goes as follows:

- When a pre-usage phase is over, the PEP starts a usage session. The PEP generates a usage session identifier and sends it with the ongoing request to the PDP;
- The PDP loads relevant policies, and builds a resulting policy using combining algorithms. The PDP analyzes the resulting policy and selects a set of mutable attributes that can change a policy state, i.e trigger access revocation, obligations or attribute updates. The PDP subscribes to the Attribute Manager be notified whenever any attribute from this set is changed. The subscription is done only once when the usage session starts.
- The Attribute Manager is always active. It handles all subscriptions and collects information about requested attributes through sensors. When a change of an attribute value occurs, the Attribute Manager pushes the attribute value to the PDP.
- The PDP wakes up and loads usage sessions related to the pushed attribute and reevaluates the policy. If no change in the access decision, the PDP awaits for the next invocation. Otherwise, the PDP performs state transition and triggers either access revocation, or attribute update, or obligation actions request to the PEP.
- The usage session is over, when either the endaccess() action is forwarded from the user, or the PDP recognizes a violation of a policy and returns “Non applicable” access decision to the PEP. Afterwards, the PDP initiates post-updates and unsubscribes from the Attribute Manager for attributes which are no longer used. All the relevant information to the usage session is deleted.

The model with active Attribute Manager is the most appropriate and intuitive for the usage control needs but it has pros and cons. The main advantage is that policy reevaluation is run immediately when a security-relevant event occurs. Unfortunately, in distributed system it is costly to discover and broadcast every attribute change from possible repositories. Frequent secure communications between authorization components provide a meaningful overhead. Further, communications between the PEP and the PDP are asynchronous and state-full. The PEP invokes the PDP only when a usage session starts or is ended by a user. The PDP calls the PEP when the access should be revoked or every time when ongoing obligations should be fulfilled. The support of the asynchronous communication requires the usage session identifier shared between the PDP and the PEP.

The alternative approach assumes *passive Attribute Manager* (pull attribute model). In this approach, fresh attributes are pulled to the PDP and the policy reevaluation in triggered periodically when a certain time interval elapsed.

Passive Attribute Manager with active PEP imposes that the PEP during the usage session periodically sends access requests to the PDP. The PDP pulls the required attributes from the Attribute Manager. The Attribute Manager questions the sensors to obtain fresh values and returns them to the PDP. The PDP makes access decision and replies to the PEP. Notice, that it is not necessary, that attribute values and/or the access decision will change between two adjacent invocations. The procedure is repeated iteratively until the usage session is ended or revoked. In this model the interactions between the PEP and the PDP are synchronous. For every request the PEP receives the response from the PDP. The PEP also manages the usage session meta-information. The current policy state is included into the access request. As the result, the PDP can be state-less and is capable to recover the policy state from the access request. After the policy reevaluation, the PDP ins erts the current policy state to the response message. The *passive Attribute Manager with active PEP* puts too much functionality on the PEP side which is usually application depended. We consider this as a limitation of the model.

Passive Attribute Manager with active PDP imposes that the PDP during the usage session periodically pulls fresh attributes values from the Attribute Manager and reevaluates the access decision. In contrast to the previous model, the PDP is active now and manages the usage session meta-information. The interactions between the PDP and the PEP are asynchronous in this case and are the same as in the case of *active Attribute Manager*.

Although the models with the passive Attribute Manager are easier to implement, they have some limitations. The most crucial issue is that a choice of a big time slot leads to a security breach. The access can not be revoked immediately when the policy is violated due to change of attributes. The access will be revoked only after the scheduled invocation. On the other side, too frequent invocations will produce a time-overhead. To minimize this shortcoming, a several adoptive checks can be used. The idea is to adapt the next scheduled policy reevaluation to the current values of the attributes depending on how far attribute values are from the critical value which triggers an obligation, or update actions, or the access revocation. As closer the values are as more frequently the policy should be reevaluated and vice versa [2]. We consider as a future work an appropriate model for the periodic policy checks.

The preferable data-flow model also depends on the attributes used in the policy. If the policy based on immutable attributes, the model with the passive Attribute Manager and active PDP is the best solution. In this case, only PDP is a responsible for all attribute updates and triggers security-relevant events. In contrast, if the policy exploits attributes mutable rarely by the PAP, environment and third parties, the model with active Attribute Manager is more appropriate.

5 A case study

We chosen an access to and usage of Grid computational services as a case study. Computational services (e.g. GRAM service in Globus Toolkit) are designed to execute remote user's applications at the resource provider side.

The traditional authorization model, where an authorization decision is taken only once before granting an access to a service, is insufficient for long-lived computational services. Initial conditions can change during the application execution due to attributes change. This requires continuous access reevaluation. The policy violation can lead to the access revocation and service termination.

We follow the XACML philosophy and impose a security policy combining many policies from distinct parties, e.g. a policy written by a VO (virtual organization) administrator and a policy written by a provider of a computational service. The VO administrator might want to implement the following policy containing 2 rules:

- Grid user should present his/her VO membership certificate with the initial request (pre-authorization). This rule refers to "preA0" model;
- No more that 10 applications in the whole VO can run on behalf of the user. If the number exceeded 10, than the access should be denied (pre-authorization). The update of the number of running applications is required before starting a service (pre-update) and when the usage is over (post-update). This rule refers to "preA13" model.

The resource provider policy contains 5 rules:

- Grid user's reputation should be higher the threshold value before the usage (pre-authorization). If it is below the threshold during the execution, the access should be revoked and the application terminated (on-authorization). After the usage, the resource provider updates Grid user's reputation (post-update). If the service was ended normally by the user, the reputation should be increased, while if the access was revoked by the system, than the reputation should be decreased. This rule refers to "onA3" model;
- No more that 5 applications can run on behalf of the user on the resource provider node. If the number exceeded 5, than the access should be denied (pre-authorization). The update of the number of running applications is required before starting a service (pre-update) and when the usage is over (post-update). This rule refers to "preA13" model;
- Grid user has to sign an agreement, e.g. that application is not malicious, before submitting it to the execution. This rule refers to "preB0" model;
- The application submitted for execution can exploit computational resources for a particular time quota. If during the usage, application's execution time is 1 hour to reach the quota value, the ongoing obligation is triggered. This obligation informs the user, that allowed execution time is elapsing and the credit is required to proceed the execution. This rule refers to "onB0" model;

- If application's execution time exceeded the quota and no credit was submitted by the user, the system should terminate the execution of the application (on-authorization). Otherwise, if the credit is presented, the system triggers attribute updates (on-update). It doubles the execution quota time limit and sets the credit value to zero. The iterative prolonging of the usage quota can be performed unbounded number of times. This rule refers to "onA2" model;

The policies given above fit to the UCON abstract model but can not be expressed in the XACML. Actually, they contain features which are not presented in the XACML, e.g. the policies have continuous checks of authorization predicates, have ongoing obligations and attribute updates. Some obligations and updates are conditioned using the application execution time attribute.

The resulting policy is the combination of the VO and the resource provider policies. Before the usage, the system first checks the VO certificate of the user, the number of running applications in the VO on the behalf of the user, the number of running applications on the local host on the behalf of the user. If these conditions are satisfied, the PDP invokes the Attribute Manager to update number of running application in the VO and on the local host. Further, the grant access and obligation to sign an agreement is sent to the PEP. The PEP enforces the obligation. If the user signs the agreement, the PEP starts the user's application execution and initializes a new usage session. During the usage, the system follows that the time quota does not exceed and a user's reputation is over a threshold. When the usage time is in hour to the time quota value, the system enforces ongoing obligation and proposes the user to obtain a credit to prolong the usage session. If the user acc epts and presents the credit, the time quota is doubled. The time quota can be prolonged unbounded number of times. Eventually, when the usage session is over, the system performs post-updates - decreases the number of running applications and updates the user's reputation appropriately.

6 Conclusions

In this paper, we introduced the U-XACML policy language which extends the XACML to capture the UCON features. We outlined the syntax and semantics and the enforcement architecture for the U-XACML. XACML has been extended to capture attribute updates, continuous checks of authorization predicates and obligations fulfilment, and conditions over ongoing attribute update and obligation actions. We studied the enforcement architecture with the event-based and a push attribute model and periodic invocation with a pull attribute model of the policy reevaluation. All approaches shown advantages and disadvantages.

Our future step will be an implementation of the U-XACML policy engine which can be exploited for the usage control over Grid resources. This work is in progress and currently we implemented the Grid service revocation mechanism [1] which can be considered as a part of the U-XACML PEP. Afterwards, we intend to give a formal model and the analysis of the U-XACML policy.

References

1. Colombo, M., Lazouski, A., Martinelli, F., Mori, P.: On Usage Control for Grid Services. In: The 2009 IEEE International Workshop on HPC and Grid Applications. Sanya, China (2009)
2. Damiani, M.L., Bertino, E., Silvestri, C.: Approach to supporting continuity of usage in location-based access control. In: FTDCS '08: Proceedings of the 2008 12th IEEE International Workshop on Future Trends of Distributed Computing Systems, pp. 199–205. IEEE Computer Society, Washington, DC, USA (2008)
3. Feng, J., Wasson, G., Humphrey, M.: Resource usage policy expression and enforcement in grid computing. IEEE/ACM International Workshop on Grid Computing pp. 66–73 (2007)
4. Hafner, M., Memon, M., Alam, M.: Modeling and enforcing advanced access control policies in healthcare systems with Sectet. In: Models in Software Engineering: Workshops and Symposia at MoDELS, pp. 132–144. Springer-Verlag, Berlin, Heidelberg (2008)
5. 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 '08: Proceedings of the 13th ACM symposium on Access control models and technologies, pp. 123–132. ACM, New York, NY, USA (2008)
6. Martinelli, F., Mori, P., Vaccarelli, A.: Towards continuous usage control on grid computational services. In: Proceedings of Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services (ICAS-ICNS 2005), IEEE Computer Society, p. 82 (2005)
7. Naqvi, S., Massonet, P., Aziz, B., Arenas, A., Martinelli, F., Mori, P., Blasi, L., Cortese, G.: Fine-Grained Continuous Usage Control of Service Based Grids - The GridTrust Approach. In: ServiceWave '08: Proceedings of the 1st European Conference on Towards a Service-Based Internet, pp. 242–253. Springer-Verlag, Berlin, Heidelberg (2008)
8. Park, J., Sandhu, R.: Towards usage control models: Beyond traditional access control. In: SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies, pp. 57–64. ACM, New York, NY, USA (2002)
9. Park, J., Sandhu, R.: The UCON_{ABC} usage control model. ACM Transactions on Information and System Security **7**(1), 128–174 (2004)
10. XACML: eXtensible Access Control Markup Language (XACML). Www.oasis-open.org/committees/xacml
11. 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 **11**(1), 1–36 (2008)