Integration of Quantitative Methods for Risk Evaluation within Usage Control Policies

Leanid Krautsevich, Aliaksandr Lazouski, Fabio Martinelli, Paolo Mori, Artsiom Yautsiukhin
Istituto di Informatica e Telematica, Consiglio Nazionale delle Ricerche
Via G. Moruzzi 1, Pisa 56124, Italy
Email: {firstname.lastname}@iit.cnr.it

***Abstract*—Usage Control (UCON) enhances traditional access control introducing mutable attributes and continuous policy enforcement. UCON addresses security requirements of dynamic computer environments like Grid and Cloud, but also raises new challenges. This paper considers two problems of usage control. The first problem arises when a value of a mutable attribute required for an access decision is uncertain. The second problem questions when to retrieve fresh values of mutable attributes and to trigger the access reevaluation during the continuous control.**

We propose quantitative risk-based methods to tackle these problems. The authorisation system grants the access if the security policy is satisfied and the risk level is acceptable. The authorisation system retrieves fresh attribute values following the strategy which minimises the risk of the usage sessions. We integrate the authorisation system based on the U-XACML language with quantitative methods for risk evaluation. We present the architecture, the implementation, and the evaluation of the overhead posed by the risk computation.

Keywords*—*Usage Control, Risk, Decision Making, Attribute Retrieval.

I. Introduction

Traditional *Access Control* models [19], such as Mandatory Access Control (MAC), Discretionary Access Control (DAC) or Role-based Access Control (RBAC), check that subjects hold the proper rights before granting them the access to the requested objects [1]. With new technologies where access sessions last for long time (such as Web Services, Cloud, Grid), it is not enough to check access rights before granting the access. Therefore further controls performed during the access sessions are required to verify that the access right is still valid. The *Usage Control (UCON)* model [13], [18], [26] was introduced to satisfy these needs.

The UCON model defines three types of policy statements: *i) authorisations*, which are the predicates over subject and object attributes; *ii) conditions*, which are predicates over environmental attributes; *iii) obligations*, which are actions which must be performed. Hence, the decision process is based on the attributes of the requesting subject, the accessed object, and the execution environment. The UCON assumes that the attributes are *mutable* and their values may change while an access is in progress. To address this issue, the UCON proposes continuous enforcement of the security policy, in order to interrupt ongoing accesses when the corresponding access rights are not valid any longer because of the new attribute values.

Getting up-to-date attribute values to perform the decision process is a crucial issue in the UCON. Attributes are collected from attribute providers and sent to the UCON authorisation system (UAS) which exploits them for the decision process [17], [20]. Some attributes (e.g., reputation and location of a subject) are *remote*, i.e., they are managed by attribute providers located outside the administrative domain of the UAS. Therefore, the attribute values may be received with delays (e.g., due to delays in delivery or processing). Moreover, in order to save resources of the attribute provider (e.g., a sensor) and bandwidth of the network, up-to-date values of the attribute are sent to the UAS periodically. In this case, some attribute values may be lost and this may lead to an undetected violation of the policy. E.g., if the UAS grants accesses only to users in a given location, a mobile user may leave such location right after the attribute check and come back right before the next attribute check [4]. Thus, there are two issues in the UCON: i) to ensure that the taken *access decision* is correct even if the attribute value is received with a delay (i.e, not up-to-date); ii) to set up the proper queries of the attribute values to ensure efficient control *during the usage session*.

This article proposes the integration of quantitative methods for risk evaluation in UCON policies. In this paper, we consider risks caused by unintentional uncertainties. First, we use risk to evaluate possible gains and losses caused by granting or denying access when the value of an attribute may be not up-to-date. Second, risk evaluation is also exploited to determine how often new values of attributes should be collected to re-evaluate the policy during the continuous control.

The main contributions of this paper are:

- an architecture for the UCON system integrated with risk service;
- an U-XACML language extended with risk functions;
- a Java-based prototype for risk-based access decisions and computation of an attribute retrieval strategy;
- an analysis of the prototype performance.

This paper is organised as follows. We start with a running example (Section II). Section III contains the mathematical basis for decision making under uncertainties and efficient attribute retrieval strategy. Section IV describes the architecture and workflow of the prototype supporting U-XACML policies. Section V is devoted to the prototype. We finish the article with related work (Sections VI) and conclusions (Sections VII).

II. Running Example

We consider a federation of Clouds studied within the EU funded project Contrail (http://contrail-project.eu) as a case-

study. We assume the following security policies expressed in natural language which govern the execution of virtual machines (VMs) in the Cloud federation:

- **Access control authorizations**: A user with a "good" or "regular" *reputation* is allowed to create and start VMs. If permitted, the user's attribute *num-vm-running* (the number of VMs running by the user) is increased. When a VM is started, its attribute *vm-srartTime* should be set to the current time.
- **Usage control authorizations**: If *num-vm-running* exceeds 5, then the UAS interrupts the VM which started first, i.e., the VM with the earliest value of *vm-srartTime*. If the overall *load* of the Cloud federation becomes "high", then the UAS interrupts all machines running on a behalf of users of a "silver" or "bronze" *category*. A user can avoid the interruption and updates the category to "gold" by paying a fee.
- **Usage control post updates**: Either the VM is interrupted by the UAS or it terminates normally, the value of the *num-vm-running* attribute must be decreased.

User's attributes (*category*, *num-vm-running*) and resource's attribute (*vm-sartTime*), needed for the evaluation of this policy, are under direct control of the UAS. Though, there are some mutable attributes, i.e., a user *reputation* and a system *load*, which values are uncertain.

For the *reputation* attribute we assume that every Cloud provider participating in the Cloud federation collects the information about execution of user's VMs on its resources and sends feedback to the central reputation management system of the federation once in 6 hours. The Cloud federation combines all information and produces the *reputation* attribute.

Assume, the UAS gets an access request from a user with "regular" *reputation* to start a VM. The problem is that if the last update in the reputation management system was 5 hours ago, then the reputation of the user might have changed since that time. Some Cloud provider may already have the information about the user which turns the reputation value to "malicious", however this information to be shared only in an hour. Thus, using the stale reputation value the UAS grants the access to the unauthorised user. To avoid such a harsh decision, we propose to estimate possible effects of granting and denying the access depending on the time elapsed since the last observation of the attribute value.

For the *load* attribute, the Cloud federation should poll every Cloud provider to estimate the overall load of the federation. In fact, frequent checks of the load require some computational power and bandwidth consumption and cannot be done very often. The UAS should balance between the frequency of checks and the potential violation of the security policy caused by unnoticed changes of the federation load.

III. Quantitative Risk Methods

In this section we recall quantitative methods for enforcement of UCON policies initially presented in [11], [12]. These methods are combined with usual mechanisms for policy enforcement. We consider the following unintentional uncertainties:

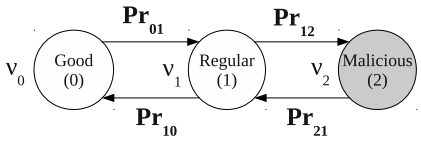

Fig. 1. CTMC for Mutable Attribute

- *freshness 1 (delays in processing)* assumes that there are inevitable time delays in delivery and processing of attribute values;
- *freshness 2 (non-continuous checks)* denotes the case when some "intermediate" changes of a mutable attribute are missed because the attribute retrieval process is carried out through some time interval.

Suppose there are several changes of an attribute value during a time interval. The values that completely describe all the changes are *real* attribute values. Usually, real values are possessed by the Attribute Manager (AM) which is not controlled by the UAS. The UAS enforces policies querying only a part of real values, called *observed* values.

Indeed, real changes of an attribute can be modelled as a *stochastic process*. In [11], the mutable attribute is represented as a discrete-time Markov chain (DTMC). Each value from the domain of the attribute corresponds to a state of the DTMC and the transition matrix defines possible changes of the attribute. Continuous attributes can be modelled in a similar way with continuous-time Markov chain (CTMC).

Example 1: Figure 1 shows the CTMC for the user's *reputation* attribute. The UAS gets the matrix $\mathbf{Prob}$ of possible one-time transition probabilities from statistics:

$$\mathbf{Prob} = \begin{pmatrix} 0.0 & 1.0 & 0.0 \\ 0.3 & 0.0 & 0.7 \\ 0.0 & 1.0 & 0.0 \end{pmatrix} \quad (1)$$

The UAS also gets rate parameters of the CTMC:

$$\mathbf{N} = \begin{pmatrix} 0.25 & 0.5 & 0.2 \end{pmatrix} \quad (2)$$

A. Risk for Access Decision Making

Suppose a subject asks for an access to an object at time t_{try}. The UAS queries a fresh value of an attribute and makes the access decision at the time t_{perm}. The UAS grants the access *correctly* at t_{perm} if observed value at t_{try} and real value at t_{perm} satisfy the policy. If the policy is violated at any of these moments the access is denied.

The correct enforcement of access control is not always possible, since there is an inevitable delay $\delta t = t_{perm} - t_{try}$ between the attribute query and the decision (*freshness 1*). The value may change several times during δt that may cause the incorrect decision. In this case, we propose the risk-based access decision making under uncertainties:

1) The UAS evaluates the policy using observed attribute values.
2) If the observed values satisfy the policy, the UAS estimates the probability $\mathbf{Pr}^{perm}$ that real attribute values satisfy the policy too. If the probability is acceptable, the UAS grants the access.

To evaluate the utility of a single access decision, we assign utilities to all possible decision outcomes. The outcomes are: *grant access when policy holds* which is evaluated with a positive utility (gain C^{GH}); *grant access when policy is violated* which is assessed with a negative utility (loss C^{GV}); *deny access when policy holds* (loss C^{DH}); *deny access when policy is violated* (gain C^{DV}). We also denote the utility (loss) of an attribute retrieval as C^a. Similar to other risk-based methods (e.g., [21]) we assume that utility values could be found using existing organisational documentation or interviews with system and information owners.

We assume that the UAS knows the statistical behaviour of the attribute (e.g., parameters of the CTMC which models the attribute). If the UAS observes the attribute value at t_{try}, it can compute $\mathbf{Pr}^{perm}$. The *risk* of making an erroneous access decision is $\mathbf{Pr}^{perm} \cdot C^{GV} + \mathbf{Pr}^{perm} \cdot C^{DH}$, while the benefit is $\mathbf{Pr}^{perm} \cdot C^{GH} + \mathbf{Pr}^{perm} \cdot C^{DV}$. The UAS compares the risk and benefit and makes the most profitable decision. The UAS grants the access if $\mathbf{Pr}^{perm}$ is greater than or equal to the threshold th, where:

$$th = \frac{C^{GV} - C^{DV}}{C^{DH} + C^{GV} - C^{DV} - C^{GH}} \tag{3}$$

An average utility of a single access decision in case of the risk-based enforcement is given by:

$$\langle C \rangle_{th} = C^a + \begin{cases} \mathbf{Pr}^{perm} \cdot (C^{GH} - C^{GV}) + C^{GV} & \text{if } \mathbf{Pr}^{perm} \geq th \\ \mathbf{Pr}^{perm} \cdot C^{DH} + C^{DV} \cdot (1 - \mathbf{Pr}^{perm}) & \text{otherwise} \end{cases} \tag{4}$$

We refer the reader to [9], [10], [11] for the details on the computation of th and $\mathbf{Pr}^{perm}$. In short, to find the th we should determine the $\langle C \rangle_{th}$ for all $\mathbf{Pr}^{perm}$ (we should take an integral of $\langle C \rangle_{th}$). Then we take the first derivative of it and find th when the derivative equals to zero.

Example 2: Assume, that for our running example (see Section II) the UAS defined the following utilities: $C^{GH} = 2.5$, $C^{GV} = -5.0$, $C^{DH} = -1$, $C^{DV} = 0$, and $C^a = -1.5$.

The UAS knows that the last update was 5 hours ago and the observed value of the reputation attribute was "regular". The UAS computes that the probability to appear in any state but "malicious" after 5 hours is $\mathbf{Pr}^{perm} = 0.48$ while th is 0.59 (see Equation 3). Thus, the access decision taken by the UAS is *deny* (since $0.48 < 0.59$), and the utility of making this decision is -2.9 (see Equation 4). Although, deciding to deny the access we get negative income, this loss is lower that the loss we get allowing the access.

B. Risk for Attribute Retrieval

The UCON model requires that the policy is continuously enforced while the allowed accesses are in process. A policy reevaluation is required each time an attribute used in the policy changes its value. A UCON policy is *correctly* enforced if the access is immediately revoked when a violation happens.

Frequent attribute querying may be impossible or ineffective, thus, the queries are done through time intervals Δt. This leads to *freshness 2* uncertainty and the correct enforcement is impossible in this case. Thus, we recall the notion of risk-based attribute retrieval initially presented in [11]. The UAS:

1) Evaluates a policy and makes the decision based on the observed attribute values. If the access decision is "deny" then the UAS interrupts the access.
2) Computes when the next attribute query should be performed.
3) Waits until this time elapses and then pulls fresh attribute values and goes to the first step.

We would like to maximise the average utility of a usage session. There are 2 new utilities for usage control: (i) c^g is a positive utility (gain) per unit of time $\Delta\tau$ when all changes of real attributes satisfy the policy; (ii) c^l is the negative utility (loss) per unit of time $\Delta\tau$ when the policy is violated. The utility C^s of the usage session depends on the amounts of time during the session when the attribute value satisfies (τ^g) and violates (τ^b) the policy. Then:

$$C^s = c^g \cdot \tau^g + c^l \cdot \tau^b + C^a \cdot (n+1) \tag{5}$$

where $n+1$ is the number of attribute queries in the session.

Let τ_j^{AM} be the time when a real attribute value changes and τ_j^{UAS} is the time when the attribute value is observed by the UAS. Then, τ^g and τ^b are given by:

$$\tau^g = \sum_{j=0}^{l} \Delta\tau_j \cdot \theta(\tau_j^{AM}); \tag{6}$$

$$\tau^b = \tau_n^{UAS} - \tau_0^{UAS} - \tau^g \tag{7}$$

where l is the length of the session, $\theta(t_j^{AM}) = 1$ if the policy holds during $\Delta\tau_j$ and $\theta(t_j^{AM}) = 0$ if the policy is violated.

Let $\mathbf{Pr}^s$ be a probability that a certain session occurs. The *average* utitlity of a single usage session is a weighted sum of costs of all possible sessions:

$$\langle C \rangle_{\Delta T} = \sum_{\forall s} \mathbf{Pr}^s \cdot C^s \tag{8}$$

where s is a sample session (sequences of observed attributes) and $\Delta T = \{\Delta t_1, \Delta t_2, ..., \Delta t_n\}$ is a set of intervals between attribute queries. Then, the length of the session l is given by $l = \sum_{\Delta t \in \Delta T} \Delta t$.

Risk-based attribute retrieval implies that the UAS chooses such ΔT that maximises profit of a session:

$$\arg\max_{\Delta T} \langle C \rangle_{\Delta T} \tag{9}$$

Set ΔT may depend on a history of attribute changes or on a current value of the attribute. The type of dependency follows from the type of a stochastic process that models the attribute.

Example 3: We continue the running example. The UAS retrieves the *load* attribute during the execution of the user's VM. Without loss of generality, we assume that the *load* attribute has the domain of three values ("low", "medium", and "high") and it can be modelled with the same CTMC presented in Figure 1.

For the CTMC, the future behaviour depends only on a current value of the attribute. Thus, the set ΔT contains two intervals for the cases when the attribute is observed in states 0 and 1 (if it is observed in state 2 the access is revoked).

The UAS uses the following utilities: $c^g = 2.5$, $c^l = -5.0$, $C_a = -1.5$. After considering possible sessions and computing their probabilities, the UAS finds the best average utility $\langle C \rangle_{\Delta T} = 7.2$ per session and the time intervals $\Delta T = \{4.7, 1.5\}$. Hence, if the UAS retrieves the *load* attribute and its value is "low", the next attribute query will be sent after 4.7 hours.

Figure 6 shows the dependence of $\langle C \rangle_{\Delta T}$ on the intervals between attribute queries $\Delta T = \{\Delta t_0, \Delta t_1\}$.

IV. Prototype

A. Security Policies

Our UAS deals with the following security policies: U-XACML policies, attribute retrieval policies and risk metadata.

1) U-XACML Policy: is written in the U-XACML policy language [3], [14] which extends the XACML language [24], the widely used access control language, with constructs for usage control. This policy encodes access and usage control general rules, e.g., the policy presented in Section II.

The U-XACML language is also very extensible and can exploit user-defined functions over security attributes. We used this feature to integrate the access decision making with the risk evaluation. We introduce a boolean function `isRisky` which outputs "true" if the risk level is negligible and the access should be granted. As the input this function takes:

- Timestamped observed attributes whose values are uncertain and contribute in the risk level. A timestamped attribute is a structured data-type that is actually a *(value;time)* tuple. For the sake of simplicity, we model such structure as a single attribute, e.g.,

```
<AttrubuteValue DataType="string">
   good;2013-02-07T20:22:26.705Z</AttributeValue>
```

- The id of the accessed resource. This id is needed to allocate a utility matrix which should be used to compute the risk threshold,
- An estimated time of the enforcement of the access decision (i.e., t_{perm}). Obviously, this time should excel the current time value with the amount needed for the policy evaluation and the risk computation.

The overall access decision is combined as "deny-override". The UAS grants the access if both the logical rules and the risk function allow the access. For the policy example given in Section II, the `isRisky` function is placed in the U-XACML policy as follows:

```
<PolicySet PolicyCombiningAlgId="deny-override" ... >
 <!-- U-XACML logic rules --!>
 <Policy ... > ... </Policy>

 <!-- Risk function --!>
 <Policy ... ><Rule Effect="Deny" ... >
    <Condition>
      <Apply FunctionId="isRisky">
         <AttributeDesignator Category="subject"
                   AttributeId="reputation"/>
         <AttributeDesignator Category="resource"
                   AttributeId="resource-id"/>
         <Apply FunctionId="addTimeDuration">
            <AttributeDesignator Category="local"
                      AttributeId="decision-delay"/>
            <AttributeDesignator Category="environment"
                      AttributeId="current-time"/>
         </Apply></Apply>
      </Condition>
 </Rule></Policy>
</PolicSet>
```

We omit the complete description of the example policy due to space limitations.

2) Attribute Retrieval Policy: specifies when to collect fresh attribute values and trigger the access re-evaluation during the ongoing access. We exploit an XML-based language proposed in [14] to express these policies.

An attribute retrieval policy consists of two main elements `Target` and `Prerequisites`. The `Target` element specifies identifiers of mutable attributes to collect. The `Prerequisites` element includes a conjunctive sequence of condition elements which must be satisfied before executing the attribute retrieval. 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. The attribute retrieval policy is enforced when the access is in progress. When the conditions in the policy hold, the fresh attributes are collected and the access re-evaluation is trigged.

We propose the model of the attribute retrieval based on the quantitative methods discussed in Section III-B. We introduce a `getRiskTolerance` function embedded into the `Condition` element of the attribute retrieval policy. It outputs an estimated time when a further enforcement of an U-XACML policy becomes too risky and the UAS has to interrupt the access or to mitigate the risk, e.g., by pulling fresh attribute values. As the input the `getRiskTolerance` function consumes:

- Timestamped observed attributes whose values are not fresh and contribute in the risk level. Only last observations of the mutable attributes should be given,
- The ids of all resources where access is ongoing. These ids are needed to assign utilities of the usage control enforcement and to compute the best strategy for the attribute retrieval.

Below there is an example of a retrieval policy for the system load. The next retrieval of the load is performed when the current time is greater than the time computed by the function `getRiskTolerance`:

```
<AttributeRetrieval>
 <Target>
   <AttributeValue>load</AttributeValue>
 </Target>
 <Prerequisites>
  <Condition>
     <Apply FunctionId="dataTime-greater-then">
        <AttributeDesignator Category="environment"
                  AttributeId="current-time"/>
        <Apply FunctionId="getRiskTolerance">
           <AttributeDesignator Category="subject"
                     AttributeId="load"/>
           <AttributeDesignator Category="resource"
                     AttributeId="resource-id"/>
     </Apply></Apply>
  </Condition>
 </Prerequisites>
</AttributeRetrieval>
```

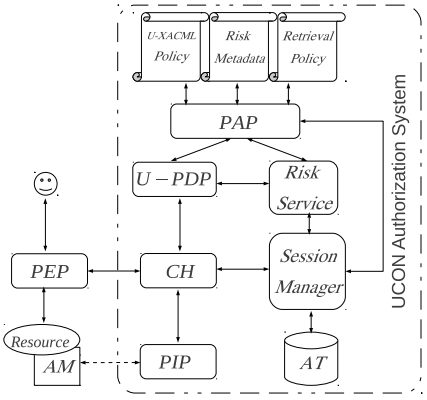

Fig. 2. UAS Architecture Integrated with Risk Service

3) Risk Metadata: contains a utility matrix for each resource and a specification of a stochastic process which models a behaviour of a mutable attribute. The utility matrix is stored in an XML file.

For the description of the stochastic properties of Markov chains, we exploit the language used in the PRISM model checker (http://www.prismmodelchecker.org). Each separate Markov chain is described within expressions `module NAME` and `endmodule` where `NAME` specifies the name of an attribute. The number of states in the chain is presented using variables. The stochastic properties of continues-time Markov chains are expressed as commands. The PRISM description of the reputation attribute:

```
ctmc
module REPUTATION
x : [0..2];
[] x=0 -> 0.25:(x'=1);
[] x=1 -> 0.15:(x'=0) + 0.35:(x'=2);
[] x=2 -> 0.20:(x'=1);
endmodule
```

Variable `x : [0..2]` denotes that the chain contains 3 states numbered from 0 to 2. Command `[] x=0 -> 0.25:(x'=1)` denotes that the transition form state 0 is possible only into the state 1 with infinitesimal transition rate of 0.25. Infinitesimal transition rates [22] for the transition from the state i into the state j are given by:

$$q_{ij} = \nu_i \cdot \mathbf{Pr}_{ij}$$

where ν_i is a rate parameter for the state i and $\mathbf{Pr}_{ij}$ is a one-time transition probability from the state i into the state j.

B. Architecture

Figure 2 shows the UAS architecture enhanced with the component for the risk evaluation. As in most authorisation systems [23], [24], the main components are:

Policy Enforcement Point (PEP) is integrated within the system hosting resources and executes security relevant actions.

Policy Information Point (PIP) provides an interface for retrieving attributes. According to the requested attributes, the PIP contacts the right **Attribute Manager (AM)** exploiting the right protocol.

Context Handler (CH) is the front-end of the UAS, that manages the protocol for communicating with PEPs and PIPs. It converts and forwards messages sent between components in the proper format.

Policy Decision Point (U-PDP) evaluates U-XACML policies for the requests received from the CH.

Risk Service (RS) is the novelty of the architecture, and it computes functions `isRisky` and `getRiskTolerance`. It is called during the decision making and when the time of the next attribute retrieval is unclear. From the architectural point of view, the nature of risk computed by `isRisky` and `getRiskTolerance` is not specified. Indeed, the RS could be used to compute other types of risks, e.g., intentional, but this is out of the scope of this paper.

Access Table (AT) keeps meta-data of accesses in progress, i.e., *usage sessions*. It contains a table of the current sessions with their statuses, a table of ids of the attributes needed to service each session, and a table of values of these attributes retrieved the last time.

Session Manager (SM) manages usage sessions. The SM creates a new entry in the AT for each new usage session. The SM also enforces the retrieval policies for mutable attributes and when the values of some attributes change, it triggers the access right re-evaluation of each usage session in the AT that exploits those attributes, until the access decision is *deny* or the sessions end normally. If two usage sessions refer to the same mutable attributes, the SM queries those attributes just once instead of sending two disjoint queries. Then, the SM triggers the access re-evaluation for both sessions with the new values.

Policy Administration Point (PAP) manages security policies.

C. Workflow

Due to space limitation, we present a message workflow only in the case when a session is revoked due to a policy violation. The natural end of access is omitted here. Figure 3 shows the message workflow between main components of the UAS architecture.

At the initialisation phase, the U-PDP loads U-XACML policies from the PAP, the SM gets retrieval policies, and the RS obtains U-XACML policies and risk metadata. For each mutable attribute, the RS computes the time of the next check (retrieval) assuming that the attribute has a given value. In fact, the RS computes an interval for every state in the Markov chain which models the attribute (see Equations 8 and 9).

The first message, *tryaccess*, is sent by the PEP to the CH when the request for the execution of a security relevant action is intercepted by the PEP. The CH retrieves the values of the attributes that could be relevant to the decision process by sending the *attr query* message to the PIP that, in turn, contacts the relevant AMs exploiting their specific protocols, and sends back these values to the CH through the message *attr value*. The CH then sends the access *request* that includes the attributes previously collected, to the U-PDP. The U-PDP evaluates the policy and calls the RS to compute the `isRisky` function. The RS computes the risk threshold and returns the decision. Then, the U-PDP combines a risk-based and a logic-based decisions and replies with the *response* to the CH.

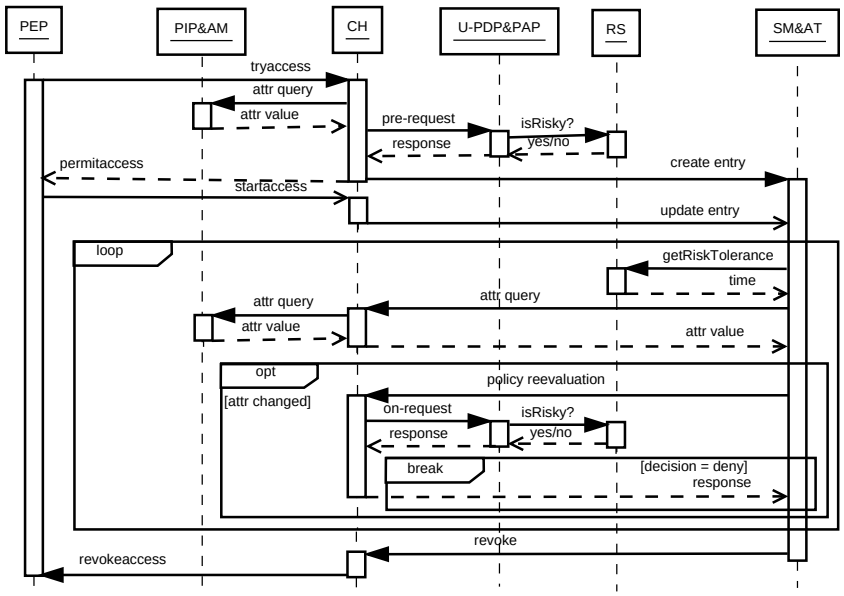

Fig. 3. Sequence Diagram of Usage Control Policy Enforcement in Case of Access Revocation

Let us suppose that the U-PDP allows the execution of the requested action, then the CH replies with the *permitaccess* message to the PEP. Before sending it, the CH sends the *create entry* to the SM for creating a new entry that represents the new usage session in the AT. Also, the *create entry* message contains attribute updates which should be performed by the SM before the usage session starts. When the access has began, the PEP sends the *startaccess* message to the CH, that sends the message *update entry* to the SM. The SM contacts the AT to change the status field of the database entry related to this usage session from *pending* to *active*, and triggers the evaluation of the ongoing access for the first time. Hence, the SM starts the continuous policy re-evaluation loop.

The SM enforces attribute retrieval policies and decides when attributes should be refreshed. The SM calls the RS to compute the `getRiskTolerance` function. For each attribute, the RS just selects an interval which corresponds to the current attribute value from a set of the intervals pre-computed at the initialisation phase. The RS adds to this interval to the current time values and replies to the SM. Then, the SM awaits until this time passes and sends the *attr query* message to the PIP to get the fresh values of attributes that are relevant for the access. The PIP gets these values from the AMs, and sends them back to the SM. If one of the collected values is different from the one cached in the AT, the SM contacts the CH sending the *policy reevaluation* message; the CH translates it in the right format and sends the *request* message to the U-PDP that performs the re-evaluation of the access right. The U-PDP evaluates the policy and calls the RS to compute the `isRisky` function the same way as during the pre-authorisation phase. If the decision included in the *response* message is *permit*, then the CH forwards this answer to the SM. The SM continues the policy enforcement by enforcing attribute retrieval policies, pulling fresh attributes, and triggering the access re-evaluation. Instead, if the response included in the *response* message sent by the U-PDP is *deny*, the SM sends the *revokeaccess* message including the data to identify the right PEP to the CH that forwards it to this PEP that will force the access revocation.

V. Implementation of the Prototype

A. UAS as a Service

The UAS is a web service based implementation of the architecture given in Fig. 2. We exploit two web-services: the PIP service which implements the functionality of the PIP, and the Authorisation Service (AS) which implements the rest. We used the Apache Axis2 framework (http://axis.apache.org/axis2) to implement and deploy these services.

The operations that can be performed by the AS correspond to the *tryaccess*, *startaccess*, and *endaccess*. The Message Exchange Pattern for the *tryaccess* is In-Out, and In-Only for other operations. The CH processes incoming and outgoing messages, i.e., access requests and responses, attribute queries. These messages are compliant with the "SAML 2.0 profile of XACML". For the CH implementation we used OpenSAML2.0 extension library (http://www.bccs.uib.no/~hakont/SAMLXACMLExtension).

The U-PDP implementation extended the Sun's XACML engine (http://sunxacml.sourceforge.net) with possibility of enforcement of U-XACML policies. Also, the Sun's XACML engine permits to plug-in new functions and make them available for any policy to use. Hence, the RS implemented the `Function` interface and provided a new function `isRisky`. We have developed our own implementation of quantitative methods for the risk evaluation. In the future, we plan to outsource computations related to Markov chain primitives to the PRISM model checker (http://www.prismmodelchecker.org).

New instances of CH, U-PDP, and RS are created per every access re-evaluation, while there is only one instance of the SM for all concurrent sessions. The SM is started by the CH when the U-PDP of the first usage session responses with the permit after evaluation of the *tryaccess*. The SM dies when the last

usage session ends or it should be revoked. The SM keeps information regarding sessions in the AT implemented as the MySQL Database. The long-standing AS with the attributes shared among the concurrent sessions was possible due to the support of the state-full web services in Axis2.

At this stage, we do not provide an engine for the evaluation of any attribute retrieval policy. We leave it for the future work. Instead, in the current implementation the SM retrieves attributes based on the results returned by the `getRiskTolerance` function which is a part of the RS. We exploited BOBYQA optimiser from apach.common. math module to pre-compute time intervals used by the `getRiskTolerance` function (Equation 9).

The PIP is a stateless web service. It receives SAMLAttributeQueries and responses with SAMLAttributeStatements.

B. Performance Evaluation

We tested the performance of our UAS on a workstation running Fedora 17 (Linux Kernel 3.7.3-101) with Intel Core 2 Duo P8600 2.40GHz and 4Gb RAM. Firstly, we evaluated the performance of the `isRisky` function. The overhead t_{all} posed by the access decision making is the sum of the following time intervals: t_{attr}, t_{pdp}, and $t_{isRisky}$. The time t_{attr} is needed to build a SAML Attribute Query, to retrieve fresh attributes from PIP, to construct the final XACML request combining the access request and the attributes received from the PIP. The time t_{pdp} is needed to evaluate an XACML request against the U-XACML and get an XACML response. The time $t_{isRisky}$ is spent for the computation of the `isRisky` function. We measured t_{all} varying the number of attributes needed by the U-XACML policy from 10 to 50. We considered only one attribute with an uncertain value. The domain of values varied from 2 to 10 (i.e., a number of states of CTMC modelling this attribute). Table I shows the obtained results.

TABLE I. OVERHEAD OF DECISION MAKING

Numb. of Attr.	10	20	30	40	50
t_{attr}, ms	84	85	87	88	90
t_{pdp}, ms	9	13	26	29	30
$t_{isRisky}$, ms	5/31	5/31	5/31	5/31	5/31
t_{all}, ms	98/124	103/129	118/144	122/148	125/151

We received that the attribute retrieval contributes most in the overhead t_{all}. There is a slight growth of t_{attr} with the number of attributes. It takes 5 ms to compute the `isRisky` function for the attribute with Markov chain of 3 states and 31 ms for the chain of 10 states. Empirical evaluation shows that the time $t_{isRisky}$ grows exponentially as number of state (the attribute domain) increases. $t_{isRisky}$ is of the same range as t_{pdp} for policies of 30 certain attributes and 1 mutable uncertain attribute with the domain of 10 values. There is a linear growth of t_{pdp} with the number of attributes. The time for the computation of `isRisky` function depending on the number of states in the CTMC is displayed in Figure 4.

We evaluated the performance of the `getTolerance` function. Figure 5 shows how the time of pre-computation of the `getTolerance` function depends on the number of states in the Markov chain. The computation of the `getTolerance` function takes seconds (s), e.g., 4.36 seconds are required for the chain of 3 states and 86.4 seconds for the chain of

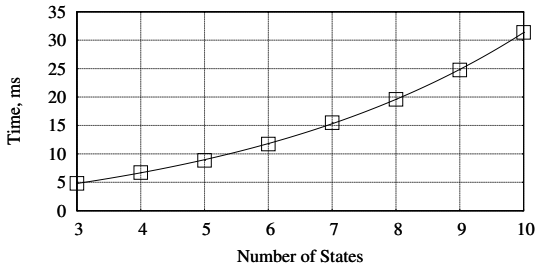

Fig. 4. Performance of `IsRisky` Function

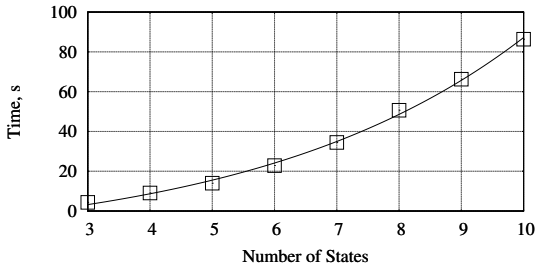

Fig. 5. Performance of `getTolerance` Function

10 states. The time of the computation grows exponentially with number of states. The exponential growth of time of computation puts limitations on the size of the domain of attribute that can be exploited within our framework. However, the `getTolerance` function should be computed only once at the initialisation phase. Then the pre-computed intervals are reused during usage control. Moreover, to provide a reasonable computational time we can exploit methods of approximation (we leave it as a future work).

The example of the average utility $\langle C \rangle$ function is presented in Figure 6. The function is built for the attribute of 3 states (see Section III). Vertical axis corresponds to $\langle C \rangle$, horizontal axes correspond to intervals δt_0 and δt_1 between checks when the attribute is observed in the state 0 in the state 1 correspondingly. The maximum average utility $\langle C \rangle = 7.2$ is received when $\delta t_0 = 4.7$ and $\delta t_1 = 1.5$.

VI. RELATED WORK

Probabilistic models are widely used for the decision making in security-related areas. Audun Jøsang proposed a beta distribution based model for subjective reasoning [8]. The model is used in reputation and trust systems. With respect to our model, the reputation and the trust can be considered as attributes of the UCON policies.

Risk-based specification of UCON policies was inspired by several articles which used risk for access control decisions [5], [15], [25]. For example, Diep et al., [5] enforced access control with risk assessment by computation of risks for access decisions taking into account various negative events and their effects on confidentiality, integrity and availability of objects. Zhang et al., [25] also considered application of risk assessment to access control but they focus on propagation of risks and benefits through a trust chain rather than to the computation of risk. In our article we focus on risks for UCON,

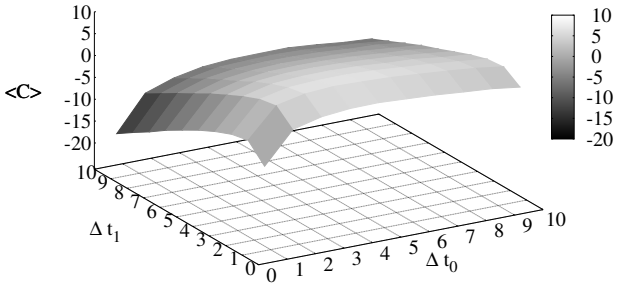

Fig. 6. Average Utility of Session Depending on Intervals between Checks

and consider the risk of making a decision with an uncertain value of an attribute.

The approach of Gheorghe et al., [7] combines security policies and attribute retrieval in a single XACML policy. In contrast, we separate U-XACML policy and attribute retrieval policy because the attribute retrieval is context-specific while the U-XACML policy expresses general access and usage rules. Several authors also paid attention to how integrate risk in access policies [6], [2]. For example, Dimmock et al. [6] extended OASIS XML policies to capture risk of an executed operation (e.g., write in a file in a Grid File Storage). In our article, we focus on usage control policies, moreover we consider a different type of risk.

The problem of attribute freshness was considered by Niu et al., [16]. The authors defined weak-stale and strong-stale safety properties and used model checking to show that the properties are satisfied. In contrast, we used a probabilistic model to make a decision even in presence of uncertainty and to ensure the maximal possible profit in average.

We presented mathematical model in details in [9], [10], [11]. In this paper, we extended our previous approaches with architecture for the enforcement of the UCON integrated with risk-based methods. We implemented the prototype and analysed the performance overhead of the proposed solutions.

VII. Conclusion

We applied risk-benefit analysis to find the most effective attribute retrieval strategy. We found that very few modifications are required in order to implement the proposed ideas in practice. For this purpose we provided required modifications of U-XACML language and enforcement architecture. We have found that the new functionality does not introduce much overhead if we consider an attribute with a small number of states. The fact, that computation of an attribute retrieval strategy requires considerable amount of time, should not affect the overall performance much since such computation is performed only once during a policy deployment. As a future work, we plan to consider risk-based decision making for policies of several attributes. Moreover, the evaluation of intentional risks (which require an attacker model) needs a more complex computational model. Finally, we are going to pay attention to the collection of statistical information about the attribute behaviour and to the evaluation of the utilities for the decision making.

VIII. Acknowledgements

This work was partly supported by EU-FP7-ICT NESSoS, EU-FP7-ICT CONTRAIL, 295354 SESAMO, and PRIN Security Horizons projects.

References

[1] M. Abadi. Logic in access control. In *Proc. of LICS-03*, IEEE.

[2] A. B. Aziz et al. Reconfiguring role based access control policies using risk semantics. *Journal of High Speed Networks*, 15(3):261–273, 2006.

[3] M. Colombo et al. A proposal on enhancing XACML with continuous usage control features. In *Proc. of CGWS-10*, Springer.

[4] M. L. Damiani et al. Approach to supporting continuity of usage in location-based access control. In *Proc. of FTDCS-08*, IEEE.

[5] N. N. Diep et al. Enforcing access control using risk assessment. In *Proc. of ECUMN-07*, IEEE.

[6] N. Dimmock et al. Using trust and risk in role-based access control policies. In *Proc. of SACMAT-04*, ACM.

[7] G. Gheorghe et al. Deploy, adjust and readjust: Supporting dynamic reconfiguration of policy enforcement. In *Proc. of Middleware-11*, Springer.

[8] Audun Jøsang. A Logic for Uncertain Probabilities. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 9(3):279–212, 2001.

[9] L. Krautsevich et al. Influence of attribute freshness on decision making in usage control. In *Proc. of STM-10*, Springer.

[10] L. Krautsevich et al. Risk-aware usage decision making in highly dynamic systems. In *Proc. of ICIMP-10*, IEEE.

[11] L. Krautsevich et al. Cost-effective enforcement of access and usage control policies under uncertainties. *IEEE Systems Journal*, 7(2):223–235, 2013.

[12] L. Krautsevich et al. Quantitative Methods for Usage Control. *Presented in QASA-12*.

[13] A. Lazouski et al. Usage control in computer security: A survey. *Comput. Sci. Rev.*, 4(2):81–99, 2010.

[14] A. Lazouski et al. A prototype for enforcing usage control policies based on xacml. In *Proc. of TrustBus-12*, Springer.

[15] Q. Ni et al. Risk-based access control systems built on fuzzy inferences. In *Proc. of ASIACCS-10*, ACM.

[16] J. Niu et al. Enforceable and verifable stale-safe security properties in distributed systems. CS-TR-2011-02, University of Texas at San Antonio, 2011.

[17] S. Osborn et al. Configuring role-based access control to enforce mandatory and discretionary access control policies. *TISSEC*, 3(2):85–106, 2000.

[18] J. Park and R. Sandhu. The UCON ABC usage control model. *TISSEC*, 7(1):128–174, 2004.

[19] P. Samarati and S. C. de Vimercati. Access Control: Policies, Models, and Mechanisms. In *Proc. of FOSAD-01*, Springer.

[20] F. B. Schneider et al. Nexus authorization logic (NAL): Design rationale and applications. *TISSEC*, 14(1):1–28, 2011.

[21] G. Stoneburner et. al. Risk management guide for information technology systems. Technical Report 800-30, NIST, 2001.

[22] H. C. Tijms. *A First Course in Stochastic Models*. Wiley, 2003.

[23] J. Vollbrecht et al. AAA authorization framework, 2000. Available via http://ietfreport.isoc.org/rfc/PDF/rfc2904.pdf

[24] XACML. eXtensible Access Control Markup Language (XACML). Availavle via www.oasis-open.org/committees/xacml.

[25] L. Zhang et al. Toward information sharing: Benefit and risk access control (barac). In *Proc. of POLICY-06*, IEEE.

[26] X. Zhang et al. Formal model and policy specification of usage control. *TISSEC*, 8(4):351–387, 2005.