Concurrent History-based Usage Control Policies

Fabio Martinelli, Ilaria Matteucci, Paolo Mori and Andrea Saracino
Istituto di Informatica e Telematica, Consiglio Nazionale delle Ricerche, Pisa, Italy

Keywords: History-based Policy, Usage Control Policy, Data Sharing, Process Algebra.

Abstract: The sharing of data and resources is one of the cornerstones of our society. However, this comes together with several challenges regarding the increasing need of guaranteeing security and privacy during both the access and the usage of such shared resources. Access control policies first, and usage control policies secondly, have been introduced to overcome issues related to the access and usage of resources. However, the introduction of distributed and cloud systems to share data and resources enables the concurrent and shared access to the same resources. Here we present an enhanced version of History-based Usage Control policies in which we are able to manage concurrent access and usage of resources by several subjects, whose actions may influence one another. Moreover, to ease the understanding of the proposed approach, we present a reference example where a document is shared among a set of people having different roles in a company.

1 INTRODUCTION

Nowadays distributed, highly connected ICT and Cloud systems are the main means to share resources (data, storage space, computational power, etc.) among organizations and/or individuals. This leads to the necessity of designing and developing proper security and privacy preserving infrastructures that manage and regulate both the access and the usage of such resources. In the recent years, several technical approaches have been proposed in order to cope with security and privacy issues in different settings (Kelbert and Pretschner, 2015; Lazouski et al., 2016). In particular, the Usage Control (UCON) model, which has been defined in (Park and Sandhu, 2004) to extend the capability of traditional access control, proved to be an effective instrument in enforcing resource security (Neisse et al., 2013; Lazouski et al., 2014). UCON enhances the expressiveness of standard access control by introducing the continuous enforcement of security policies while an access is in progress, being able to interrupt this access as soon as the policy is not satisfied any more. However, complex systems might have security requirements for which the standard UCON model is not expressive enough. To meet the security needs of such complex systems, in (Martinelli et al., 2016) the authors introduced History-based UCON policies, which allow to define the allowed behaviour of a system by combining Usage Control policies through proper operators. History-based Usage Control policies are necessary in those scenarios where the right of executing an action does not depend on that action only, but also on (a proper subset of) all actions that have been previously executed on the system.

Notwithstanding, still more work is required to deal with issues of concurrency and shared resources. In fact, in real scenarios a shared resource can be concurrently used by several actors, working in different fields and having different purposes. For example, we can consider two individuals, respectively a doctor and a patient, who are both related to a certain medical examination document. Both of them have the right of use this document according to a specific (and different) History-based Usage Control policy. This leads to the need of combining these policies in a unique History-based Usage Control policy able to describe all the behaviours allowed on the document. Hence, in this paper we propose an enhancement to the work in (Martinelli et al., 2016) by considering concurrent History-based Usage Control policies, i.e., we add the possibility of combining policies through a *process algebra interleaving* operator, named *parallel operator*. This allows us to model concurrent behaviours and the possible interdependencies that may exists among History-based Usage Control policies through process algebra operators.

The rest of the paper is structured as follows: the next section presents the scenario we refer to, and describes the importance of History-Based Usage control policies for regulating data usage in that scenario. Section 3 recalls some basic notions about the UCON

Martinelli, F., Matteucci, I., Mori, P. and Saracino, A.
Concurrent History-based Usage Control Policies.
DOI: 10.5220/0006232506570666
In *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2017)*, pages 657-666
ISBN: 978-989-758-210-3

model and the U-XACML language and Section 4 reports the preliminary notions of History-Based Usage Control policies. Section 5 presents the main contribution of this paper consisting of the enhanced version of History-based Usage Control policies. This new version is able to model and express policies to regulate the usage of resources in more complex scenarios, in which concurrent behaviours has to be combined together in a way to model also the interdependencies among them. Section 6 discusses about related work and Section 7 draws the conclusion of the paper and presents our ongoing work.

2 REFERENCE EXAMPLE

In this section we define a reference example, which will be used within the paper. Let us suppose that a company *manager M* produces some strategic documents and he wants to share them with the other *employees* through the (mobile) devices provided by his company. Each of these documents is related to a specific project of the company, thus being a valuable asset for the company itself. As a consequence, *M* needs to regulate the usage of those documents by defining and enforcing proper security policies. At first, all documents written by *M* must be validated both by one Legal and one Scientific representative of the company. These two validations can be performed in any order, even in parallel. Only after the document has been validated both from the scientific and legal points of view, it can be visualized by the employees of the company. Any subject (company representative or employee) is allowed to perform actions only on the documents related to the set of projects assigned to him by the manager. Moreover, we assume that the manager wants that each employee (E) is allowed to visualize at the same time only documents related to the same project. The set of projects assigned to each subject changes over time, depending on the company needs. Hence, the right of an employee of visualizing a given document could be revoked when he is still visualizing it. Let us suppose that an *employee E* is visualizing a set of *documents D* concerning the project $P1$. If E opens a document related to another project, say $P2$, the system should interrupt the visualization of the documents related to $P1$. The visualization of the documents related to the project $P1$ should be interrupted also when the company manager changes the set of projects assigned to the employees, and project $P1$ is not assigned to E anymore.

The previous example shows the importance of having a policy language allowing:

- to express the order in which actions can be executed. In other words, the policy should be able to state that action B can be performed only after action A.
- to express that the execution of some operations can be performed in parallel (i.e., those operation can be executed in any order, at the same time, or one operation can be started when the other is still in execution).
- to express the interdependence between the execution of different operations. In particular, the execution of one operation could cause the interruption of another, which is currently in progress.

Hence, in order to satisfy the requirements of the previous scenario, we need to define one usage control policy, UP_{LR}, which authorizes one Legal Representative (LR) of the company to perform the validation of the documents related to the projects assigned to him, one usage control policy, UP_{SR}, which authorizes one Scientific Representatives (SR) of the company to perform the validation of the documents related to the projects assigned to him, and one usage control policy, UP_E, which allows each employee (E) to visualize the documents related to one of the projects assigned to him at the same time. Moreover, we need to combine such policies with a proper language to state that UP_{LR} and UP_{SR} are enforced in parallel as soon as the document has been produced, that UP_E can be enforced only after both UP_{LR} and UP_{SR} have been terminated, and that any number of UP_E can be enforced in parallel.

3 USAGE CONTROL AND U-XACML

The UCON model (Park and Sandhu, 2004; Zhang et al., 2005) goes beyond traditional access control ones, since it takes into account attributes of users and objects which might change their value over time, i.e., *mutable attributes*. With reference to the example in Section 2, the set of projects assigned to each subject is represented in our system by a mutable attribute, which is paired with the subject. This attribute is mutable because the company can update it over time according to its needs. In particular, the value of mutable attributes could change while some actions are in progress. Consequently, the UCON model allows policy makers to write *ongoing conditions* and *obligation*, i.e., rules that must be continuously satisfied while the action is in progress. Hence, when the value of an attributes changes in a such a way that one of these ongoing rules is violated, the Usage Control policy is violated as well, and the execution of the related

action is properly suspended or interrupted. Ongoing rules satisfy a requirement of the reference example, where we need to ensure that subjects are allowed to carry on operations only on documents related to projects assigned to them for the whole duration of the operation.

The U-XACML language has been defined in (Colombo et al., 2010), and it is an extension of the XACML language (OASIS, 2013) meant to express the continuity of policy enforcement required for dealing with attribute mutability. In fact, to represent the continuity of policy enforcement, in each XACML `<Condition>` and `<ObligationExpression>`, the U-XACML language allows to specify when the evaluation must be executed by adding the clause *DecisionTime*. The conditions and obligations whose decision time is set to *pre* are evaluated at access request time, while the conditions and obligations whose decision time is set to *on* must be continuously evaluated while the access is in progress. Obligation can also be performed after the end of the access, by setting the value of *DecisionTime* to *post*.

Finally, U-XACML introduces a new element, `<AttrUpdates>`, to define attribute updates. This element includes a number of `<AttrUpdate>` elements to specify each update action. Each `<AttrUpdate>` element also specifies when the update must be performed through the clause *UpdateTime* which can have one of the following values: *pre* (the update is performed at request time), *on* (the update is performed while the access is in progress), and *post* (the update is performed when the access is terminated).

4 FORMAL SPECIFICATION OF A HYSTORY-BASED U-XACML POLICY

To be able to define the allowed behaviour of the subjects on the system, we exploit a process algebra to enhance the Usage Control model with History-based capabilities. In other words, policy makers define the set of states of the system, and they exploit process algebra like operators to define which Usage Control policy must be enforced in each of these states, and the new states resulting from the enforcement of these Usage Control policies. In this way, we satisfy another requirements of the reference example because, exploiting the history-based capabilities, the policy can state that the visualization operation can be executed only after the legal and scientific validation operations.

History-based U-XACML policies have been introduced in (Martinelli et al., 2016) and consist of U-XACML policies composed through (some of) the behavioural operators of the *PO*licy *L*anguage based on *P*rocess *A*lgebra (*POLPA* (Baiardi et al., 2004)). We chose the U-XACML language to express Usage Control policies because it directly supports all the features of the Usage Control model and it benefits from the availability of existing enforcement tools. The POLPA language, instead, describes the behaviour of entities in terms of allowed sequences of processes. The idea is to use process algebra operators to combine U-XACML policies instead of processes. This enables policy makers to have the expressiveness of the POLPA language, directly enforceable through the extended XACML, which can be evaluated through off-the-shelf standard tools[1]. It is worth noting that the approach can be easily extended to other PA-based languages, different from POLPA.

Let us assume that each Usage Control policy UP is paired with one action only, α_{UP}, which can be terminated either by the user (normal termination) or interrupted by the system (revoked) because of a policy violation, due to updates of the values of mutable attributes. For instance, the policy UP_E of the reference example regulates the action $\alpha_{UP_E} =$ VISUALIZE, that can be performed by any employee of the company. Indeed, UP_E terminates with *endAccess* if the employee explicitly ends the visualization of the considered document, i.e., the execution of α_{UP_E} terminates normally. Instead, α_{UP_E} is terminated by the system with *revokeAccess*, if, for instance, the set of projects assigned to the employee E changes and the documents that E is visualizing belong to projects no longer assigned to E. Hence, the right to executed α_{UP_E} is revoked by the system as a consequence of a change in the access context. We model these conditions with the predicate $exit(UP, r)$, that holds if the policy UP has been enforced with result r ($r \in \{endAccess, revokeAccess\}$). Hence:

$$\alpha ::= \varepsilon \parallel \alpha_{UP} \parallel \alpha_{UP}^{ea} \parallel \alpha_{UP}^{ra} \qquad (1)$$

where ε denotes no actions, α_{UP} denotes the action α associated to the policy UP, α_{UP}^{ea} is α_{UP} that correctly ends, i.e., no policy violations occur during the execution of the action, and α_{UP}^{ra} denotes that the execution of α_{UP} is interrupted by the system because of a policy violation. Instead, we use α_{UP} when who terminates the action (the user or the system) is immaterial in the policy.

A History-based U-XACML policy, hereafter denoted by HUP, results from the composition of U-XACML policies and other History-based U-

[1] http://xacmlinfo.org/category/balana/

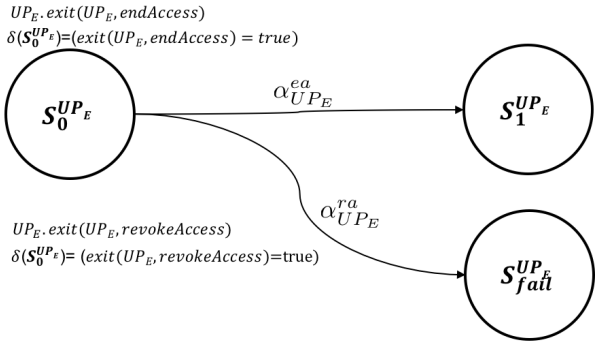

Figure 1: $\delta - LTS$ of the Usage control Policy UP_E of the Reference Example.

XACML policies according to the following grammar:

$$HUP ::= 0 | UP | UP.exit(UP, r) | HUP_1; HUP_2 | HUP_1 or HUP_2 \quad (2)$$

The informal semantics is the following:

- 0 denotes that there are no more policies to enforce;
- UP denotes the basic U-XACML policy;
- $UP.exit(UP, r)$ is the basic policy followed by the predicate *exit* stating the result of the enforcement of UP (specified by r, where $r \in \{endAccess, revokeAccess\}$). The evaluation of $exit(UP, r)$ depends on the function δ that works as follows: If UP permits the execution of an action, and this action normally terminates (α^{ea}_{UP}), then $\delta(UP) = endAccess$. Instead, if this action is interrupted because of a policy violation (α^{ra}_{UP}), then $\delta(UP) = revokeAccess$. Note that, when the evaluation UP by δ does not match the policy requirement, *e.g.*, $\delta(UP) = revokeAccess$ and the policy is $UP.exit(UP, endAccess)$, we assume that no action is performed (ε) and the policy to be enforced does not change.
- $HUP_1; HUP_2$ is the *sequential operator*. It represents the possibility of behaving as HUP_1 and then as HUP_2. Note that, both HUP_1 and HUP_2 are composed by a finite number of HUP_i^1 and HUP_j^2, where $i, j \in I$ is a finite set of indexes.
- $HUP_1 or HUP_2$ is the *choice operator*. It represents the non deterministic choice between HUP_1 and HUP_2. Hence, $HUP_1 or HUP_2$ chooses to behave either as HUP_1 or HUP_2 in a non deterministic way.

Remark 4.1. *It is worth noting that each basic usage control policy UP can be written as $UP.exit(UP, endaccess)$ or $UP.exit(UP, revokeaccess)$. Indeed, the composition through the choice operator of the two possible exit values of the policy has the same behaviour of the UP policy without explicitly express how it ends. Due to the one-to-one relation between UP and α_{UP}, this is equivalent to state that $\alpha_{UP} = (\alpha^{ea}_{UP}$ or $\alpha^{ra}_{UP})$ (where we exploited the same grammar defined in (2) to combine actions).*

Each policy HUP can be modeled through a $\delta - LTS$ (Labeled Transition system).

Definition 4.1 ($\delta - LTS$ for a HUP)**.** *Let HUP composed by a finite number of HUP_i. $\mathcal{M} = (S, Act, \mathcal{T}, \delta)$ is a $\delta - LTS$ modelling HUP, where*

- *$S = \{s | s_0 \cup \bigcup_{i \in I} S_{HUP_i}\}$, s_0 is the initial state;*
- *Act is the set of security relevant actions α of the HUP. We consider for each action α, four labels: ε, denoting no action, α_{UP_i} denotes that the action has been terminated, $\alpha^{ea}_{UP_i}$ denotes that the action has been terminated by the user, while $\alpha^{ra}_{UP_i}$ denotes that the action has been revoked by the system;*
- *$\mathcal{T} \subseteq S \times Act \times S$ is the transition relation, driven by the rules defined in Table 1.*
- *$\delta : S \to \{endAccess, revokeAccess\}$ is a labeling function that associates the value of the exit condition to the UP_i enforced in that state. In practice, it is the enforcement decision function that, by evaluation of the access request, enforces the usage policy UP_i during the execution of the action in order to evaluate if it terminates correctly or it is revoked.*

For instance, the $\delta - LTS$ that graphically represents the policy UP_E is the one depicted in Figure 1.

Each History-based Usage Control policy specifies its scope, which defines to which entity the state refers to. In particular, we define three distinct scopes: SUBJECT, OBJECT, or GLOBAL.

If the scope is SUBJECT, each subject of the scenario has his own state, and distinct subjects are paired to distinct states. Hence, when a subject s tries to perform an action, the system takes into account the state paired with s to select the set of Usage Control policies to be enforced, and updates this state as

Table 1: Semantics rules for inferring the admissible behaviour of HUP.

Basic Case.

$$\frac{}{UP \xrightarrow{\alpha_{UP}} 0}$$

endAccess.

$$\frac{\delta(UP) = endAccess}{UP.exit(UP, endAccess) \xrightarrow{\alpha^{ea}_{UP}} 0} \qquad \frac{\delta(UP) = revokeAccess}{UP.exit(UP, endAccess) \xrightarrow{\varepsilon} UP.exit(UP, endAccess)}$$

revokeAccess.

$$\frac{\delta(UP) = endAccess}{UP.exit(UP, revokeAccess) \xrightarrow{\varepsilon} UP.exit(UP, revokeAccess)}$$

$$\frac{\delta(UP) = revokeAccess}{UP.exit(UP, revokeAccess) \xrightarrow{\alpha^{ra}_{UP}} 0}$$

Prefix.

$$\frac{HUP_1 \xrightarrow{\alpha} HUP_1'}{HUP_1; HUP_2 \xrightarrow{\alpha} HUP_1'; HUP_2} \qquad \frac{HUP_2 \xrightarrow{\alpha} HUP_2'}{0; HUP_2 \xrightarrow{\alpha} HUP_2'}$$

Choice.

$$\frac{HUP_1 \xrightarrow{\alpha} HUP_1'}{HUP_1 or HUP_2 \xrightarrow{\alpha} HUP_1'} \qquad \frac{HUP_2 \xrightarrow{\alpha} HUP_2'}{HUP_1 or HUP_2 \xrightarrow{\alpha} HUP_2'}$$

a consequence of the action. Thus, the current state paired to subject *s* depends on the actions that *s* performed on the objects of the system.

Instead, if the policy scope is OBJECT, each object of the scenario has its own state. In this case, when a subject wants to access an object *o*, is the state paired with *o*, which determines the set of Usage Control policies that must be enforced. The action performed by any subject on the object *o* results in an update of the state paired to *o*.

Finally, if the scope is GLOBAL the state is shared, i.e., the actions performed by all the subjects on all the objects affect the same state.

5 CONCURRENT HYSTORY-BASED U-XACML POLICY

To fulfil the requirements defined in Section 2, in this paper we extend the History-based U-XACML policy language introduced in (Martinelli et al., 2016) and recalled in Section 4, in which the process-algebra like operators defined by the POLPA language are exploited to combine Usage Control policies expressed with the U-XACML policy language. In particular, in this paper we introduce the parallel operator, which allows to express the behaviour of a system through the concurrent application of different policies.

The parallel composition of two History-based Usage Control policies HUP extends the grammar in (2) and it is expressed as follows:

$$HUP ::= HUP \|_L HUP$$

where L is a subset of Act, the set of security relevant actions, on which the two policies synchronize their behaviour. In particular, the parallel operator allows the two policies to proceed in parallel and eventually coordinate on the set of actions L. Furthermore, we need to refine Definition 1 of actions $\alpha \in Act$ by introducing α^{sa}_{UP} as follows:

$$\alpha ::= \varepsilon \parallel \alpha_{UP} \parallel \alpha^{sa}_{UP} \parallel \alpha^{ea}_{UP} \parallel \alpha^{ra}_{UP} \qquad (3)$$

where α^{sa}_{UP} denotes the beginning of the action α_{UP}, which is called *startAccess* in Usage Control. Hence, each action α_{UP} is first started (α^{sa}_{UP}), and then terminated either by the user (α^{ea}_{UP}), or it is interrupted by the system (α^{ra}_{UP}). This helps us to be more precise in modeling the behavior of concurrent usage control policies that may start to be evaluated concurrently.

Recalling the example and Remark 4.1, the policy UP_E is:

$$\begin{array}{rcl} UP_E & = & UP_E.exit(UP_E, endaccess) \\ & or & UP_E.exit(UP_E, revokeaccess) \end{array}$$

since, both $UP_E.exit(UP_E, endaccess)$ and $UP_E.exit(UP_E, revokeaccess)$ are now refined as follows:

$$UP_E.exit(UP_E, endaccess) = \alpha^{sa}_{UP_E}; \alpha^{ea}_{UP_E}$$

and:

$$UP_E.exit(UP_E, revokeaccess) = \alpha^{sa}_{UP_E}; \alpha^{ra}_{UP_E}$$

Table 2: Refined *endAccess* semantics rule.

$$\frac{\delta(UP) = endAccess}{UP.exit(UP, endAccess) \xrightarrow{\alpha^{sa}_{UP};\alpha^{ea}_{UP}} 0}$$

$$\frac{\delta(UP) = revokeAccess}{UP.exit(UP, endAccess) \xrightarrow{\varepsilon} UP.exit(UP, endAccess)}$$

Table 3: Parallel operator semantics rule.

$$\frac{HUP_1 \xrightarrow{\alpha_{UP}} HUP_1'}{HUP_1 \|_L HUP_2 \xrightarrow{\alpha_{UP}} HUP_1' \|_L HUP_2} \quad \alpha_{UP} \notin L$$

$$\frac{HUP_2 \xrightarrow{\alpha_{UP}} HUP_2'}{HUP_1 \|_L HUP_2 \xrightarrow{\alpha_{UP}} HUP_1 \|_L HUP_2'} \quad \alpha_{UP} \notin L$$

$$\frac{HUP_1 \xrightarrow{\alpha_{UP}} HUP_1' \; HUP_2 \xrightarrow{\alpha_{UP}} HUP_2'}{HUP_1 \|_L HUP_2 \xrightarrow{\alpha_{UP}} HUP_1' \|_L HUP_2'} \quad \alpha_{UP} \in L$$

then:

$$UP_E = \alpha^{sa}_{UP_E};\alpha^{ea}_{UP_E} \; or \; \alpha^{sa}_{UP_E};\alpha^{ra}_{UP_E} \tag{4}$$

This leads to a further refinement of the semantics rules describing the *Basic case*, *endAccess*, and *revokeAccess* case given in Table 1. For instance, the *endAccess* rule is refined as in Table 2.

It is worth noting that the refined definition of UP_E in Equation 4 can be written as a deterministic process, as it is represented in Figure 2:

$$UP_E = \alpha^{sa}_{UP_E};(\alpha^{ea}_{UP_E} \; or \; \alpha^{ra}_{UP_E}) \tag{5}$$

As usual for process description languages, other operators may be derived. By using the constant definition, the sequence and the derived parallel operators, the iteration and replication operators, $\texttt{it}^n(HUP)$ and $\texttt{rec}(HUP)$ resp., can be derived. Informally, $\texttt{it}^n(HUP)$ behaves as the iteration of HUP n times, where n can also be zero, $(HUP;HUP;\ldots;HUP)$, while $\texttt{rec}(HUP)$ is the parallel composition of the same process an unbounded number of times $(HUP\|_\emptyset HUP\|_\emptyset \ldots \|_\emptyset HUP)$.

Recalling the Reference Example. With reference to the example described in Section 2, we define the following usage control policies:

- UP_{SR} is the usage control policy regulating the execution of the scientific validation of a document by a Scientific Representative of the company (the U-XACML representation of UP_{SR} is shown as an example in Table 5).
- UP_{LR} is the usage control policy regulating the execution of the legal validation of a document by a Legal Representative of the company.
- UP_E is the usage control policy regulating the visualization of a document by an employee of the company.

In order to satisfy the requirements of the reference example, we define a History-based usage control policy, HUP, by combining the previous policies through the operators of the POLPA language described in this section and in Section 4. Since we are interested in regulating the usage of each document by any of the subjects of the scenario, the scope of HUP is set to OBJECT. Consequently, each document has its own state, which is represented by a mutable attribute of the object itself, and each action performed on this document by any subject of the scenario changes this state. In this way, HUP states that the usage control policies UP_{SR} and UP_{LR} are enforced in parallel (i.e., the scientific and the legal validation can be performed concurrently, started in any order), and that the policy UP_E is enforced only after the scientific and legal validations have successfully terminated. Table 4 shows the History-based U-XACML policy for the reference example.

The first two lines of HUP (lines l1 and l2) concern the scientific validation of the document. In particular, the policy rule in line l2 states that the Scientific Representative of the company is allowed to perform the validation of the document. This rule is combined through the sequence operator with the rule in line l1, which states that the scientific validation process can be started by a Scientific Representative and then revoked by the system any number of times (represented by n in the policy), before being successfully executed, i.e., terminated by the user $((UP_{SR}.exit(UP_{SR}, endAccess)))$. The value of n is equal to 0 when the first attempt of scientific validation is carried out successfully. The rules in lines l4 and l5 are equivalent to the rules in lines l1 and l2. The only difference is that they concern the legal validation which can be performed by a Legal Representative of the company. The parallel operator in line l3 is aimed at allowing the concurrent execution of the scientific validation (lines l1-l2) and the legal validation (lines l4-l5).

Finally, the sequential operator in line l6 is aimed at allowing the visualization of the document by employees only after the termination of the parallel execution of the scientific and legal validations (l1-l5). Moreover, the document visualization allowed by the usage control policy UP_E can be performed in parallel any number of times, since UP_E is included in a replication operation $(rec(UP_E))$.

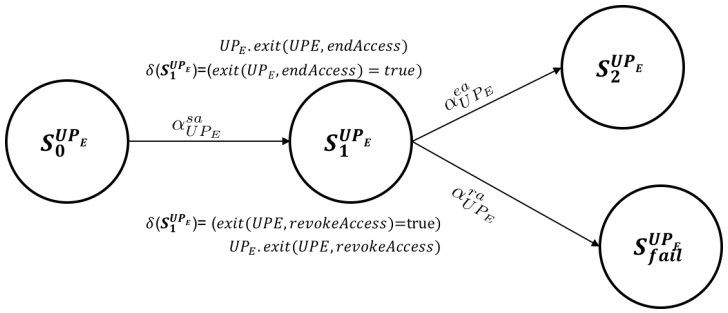

Figure 2: $\delta - LTS$ of the Usage control Policy UP_E in Equation 5.

Table 4: History-based Usage Control Policy for the Reference Example.

*l*1	$HUP =$	(	(	$\texttt{it}^{\texttt{n}}(UP_{SR}.exit(UP_{SR}, revokeAccess))$;
*l*2				$(UP_{SR}.exit(UP_{SR}, endAccess)))$
*l*3			$\|_\emptyset$	
*l*4			(	$\texttt{it}^{\texttt{m}}(UP_{LR}.exit(UP_{LR}, revokeAccess))$;
*l*5				$(UP_{LR}.exit(UP_{LR}, endAccess)))$
*l*6		);		
*l*7				$rec(UP_E)$

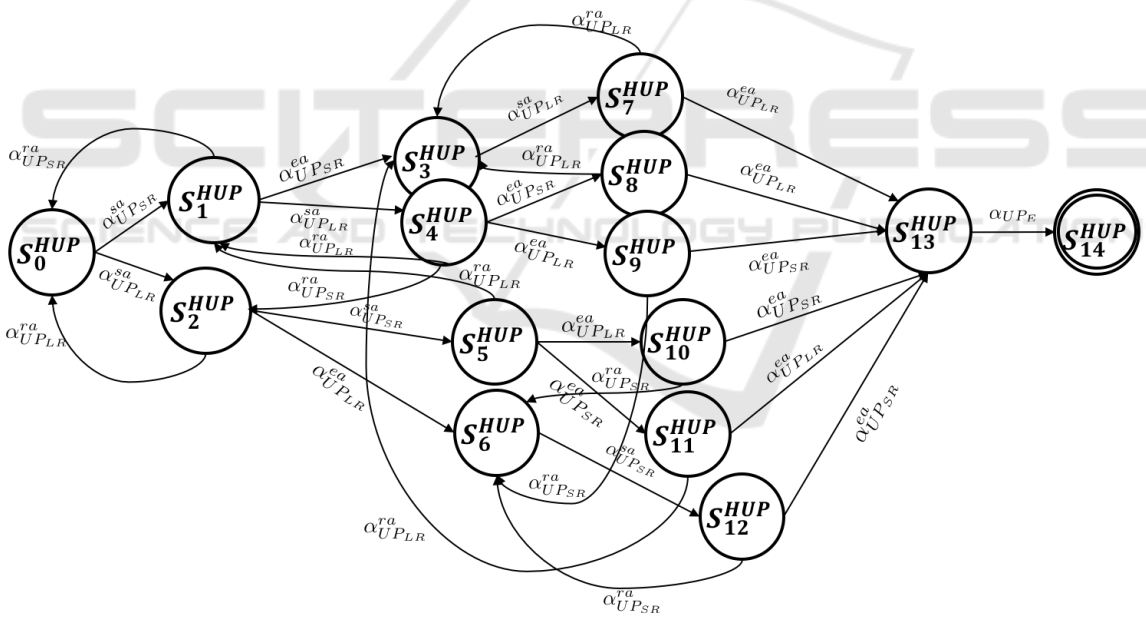

Figure 3: $\delta - LTS$ of the Usage control Policy HUP of the Reference Example.

Figure 3 represents the δ-LTS of the Hystory Based Usage Control Policy specified in Table 4. For the sake of readability, in the figure we omit to explicitly describe the conditions on states that consent the transition from a state to another with a specific action.

It is worth to specify that the proposed model does not consider deadlock issues or critical race conditions. In fact, such issues are not related directly to the model, but to the policy design. A badly written policy can indeed bring non-managed critical race conditions, which might be object of investigation in future work.

6 RELATED WORK

First definition of the Usage Control model has been proposed by Sandhu et al. in (Park and Sandhu, 2004), whilst a first application on collaborative computing system is discussed in (Zhang et al., 2008). A recent work on application of Usage Control for handling ongoing access on data has been proposed in (Lazouski et al., 2016). The proposed framework has been designed and implemented for Android devices, partially leveraging on native security mechanisms and the presence of a TPM for integrity management. This work is based on standard Usage Control, i.e., does not leverage history-based policies, in an environment such as smartphones where the higher expressiveness of History-Based policies, could bring a noticeable benefit.

In (Kelbert and Pretschner, 2014) Kelbert and Pretschner present an application of Usage Control to distributed and multi domain systems. In the presented work, the authors assume that data can travel across different domains where the same policy has to be enforced. Also this application can benefit from the extension to Usage Control presented in the current work, defining conditions where usage is granted only if the specific data has been first opened and then archived by a specific sequence of agents of the distributed system. From the same authors, another framework which is specific for data Usage Control is presented in (Kelbert and Pretschner, 2015). This framework offers a decentralized and distributed enforcement infrastructure, which however does not consider and enforce history-based policies.

On the formal aspects, the ConSpec language, presented in (Aktug and Naliuka, 2008) is another language which can express history-based policies. ConSpec can be expressed either as a labeled transition system or in a text form. However, the ConSpec language is not compliant with standards, sharing thus the same strength and weaknesses shown in the POLPA language.

The POLPA language has been previously adopted in (Martinelli and Mori, 2007) for improving the Java native security support by enabling the enforcement of history-based access control policies. Instead, a proposal of history-based Usage Control system entirely based on the POLPA language is presented (Martinelli and Mori, 2010). However, writing policies in POLPA dealing also with Usage Control features, is not straightforward.

The approach proposed in this paper, which proposes to write the Usage Control policies in U-XACML and exploits POLPA to combine them, eases the work of policy makers.

7 CONCLUSION

In this paper we extended the formal approach presented in (Martinelli et al., 2016) which defines a History based U-XACML Usage Control policy language by combining U-XACML policies through the operators of the POLPA language. The proposed extension to the language allows the enforcement of two (or more) Usage Control policies in parallel, i.e., the related operations can be executed in any order and an operation can be started when the other(s) is (are) still in execution. We also presented a motivating example, where a company needs to regulate the usage of a document shared among a set of employees, and we show that the proposed language can be successfully exploited to define the policy which encodes the set of sharing requirements.

As ongoing work, we are implementing the proposed framework to validate it and evaluate it in terms of its performance.

ACKNOWLEDGEMENTS

This work was partially supported by the H2020 EU funded project *NeCS* [GA #675320], by the H2020 EU funded project *C3ISP* [GA #700294], and by the EIT Digital High Impact Initiative *#14605 Trusted Data Management with Service Ecosystem*.

REFERENCES

Aktug, I. and Naliuka, K. (2008). ConSpec - A formal language for policy specification. *Science of Computer Programming. Special Issue on Security and Trust*, 74(1):2 – 12.

Baiardi, F., Martinelli, F., Mori, P., and Vaccarelli, A. (2004). Improving grid services security with fine grain policies. In *On the Move to Meaningful Internet Systems 2004: Confederated International Workshops and Posters, GADA, JTRES, MIOS, WORM, WOSE, PhDS, and INTEROP 2004, Agia Napa, Cyprus, October 25-29, 2004. Proceedings*, pages 123–134.

Colombo, M., Lazouski, A., Martinelli, F., and Mori, P. (2010). A proposal on enhancing xacml with continuous usage control features. In *Grids, P2P and Services Computing*, pages 133–146, Boston, MA. Springer US.

Kelbert, F. and Pretschner, A. (2014). Decentralized distributed data usage control. In *Cryptology and Network Security: 13th International Conference, CANS 2014, Heraklion, Crete, Greece, October 22-24, 2014. Proceedings*, pages 353–369, Cham. Springer International Publishing.

Kelbert, F. and Pretschner, A. (2015). A fully decentralized data usage control enforcement infrastructure. In *Applied Cryptography and Network Security: 13th International Conference, ACNS 2015, New York, NY, USA, June 2-5, 2015, Revised Selected Papers*, pages 409–430, Cham. Springer International Publishing.

Lazouski, A., Martinelli, F., Mori, P., and Saracino, A. (2014). Stateful usage control for android mobile devices. In *Security and Trust Management - 10th International Workshop, STM 2014, Wroclaw, Poland, September 10-11, 2014. Proceedings*, pages 97–112.

Lazouski, A., Martinelli, F., Mori, P., and Saracino, A. (2016). Stateful data usage control for android mobile devices. *International Journal of Information Security*, pages 1–25.

Martinelli, F., Matteucci, I., Mori, P., and Saracino, A. (2016). Enforcement of U-XACML history-based usage control policy. In *Security and Trust Management - 12th International Workshop, STM 2016, Heraklion, Crete, Greece, September 26-27, 2016, Proceedings*, volume 9871 of *Lecture Notes in Computer Science*, pages 64–81. Springer.

Martinelli, F. and Mori, P. (2007). Enhancing java security with history based access control. In *Foundations of Security Analysis and Design IV*, pages 135–159. Springer-Verlag.

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

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

OASIS (2013). eXtensible Access Control Markup Language (XACML) Ver. 3.0.

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

Zhang, X., Nakae, M., Covington, M. J., and Sandhu, R. (2008). Toward a usage-based security framework for collaborative computing systems. *ACM Transactions on Information and System Security*, 11(1):3:1–3:36.

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

Table 5: Usage Control Policy UP_{SR}.

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xacml="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
  xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17
  http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd" PolicyId="UP_SR"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable"
  Version="3.0">
  <Target/>
  <Rule Effect="Permit" RuleId="rule1">
    <Target>
      <AnyOf>
        <AllOf>
          <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Validate
            </AttributeValue>
            <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
              Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
              DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true">
            </AttributeDesignator>
           </Match>
        </AllOf>
      </AnyOf>
    </Target>
    <Condition DecisionTime="pre">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
            <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:3.0:subject:role"
              Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
              DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true">
            </AttributeDesignator>
          </Apply>
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
          ScientificRepresentative</AttributeValue>
        </Apply>
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
            <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:3.0:subject:assigned-proj"
              Category="urn:oasis:names:tc:xacml:3.0:attribute-category:access-subject"
              DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true">
            </AttributeDesignator>
          </Apply>
          <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
            <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:3.0:resource:project"
              Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
              DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true">
            </AttributeDesignator>
          </Apply>
        </Apply>
      </Apply>
    </Condition>
    <Condition DecisionTime="on">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
          <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:3.0:subject:assigned-proj"
            Category="urn:oasis:names:tc:xacml:3.0:attribute-category:access-subject"
            DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true">
          </AttributeDesignator>
        </Apply>
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
          <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:3.0:resource:project"
            Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
            DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true">
          </AttributeDesignator>
        </Apply>
      </Apply>
    </Condition>
  </Rule>
</Policy>
```