CATT: A Cloud Based Authorization Framework with Trust and Temporal Aspects

Ehtesham Zahoor
National University of Computer and Emerging Sciences, NUCES, Islamabad, Pakistan
{ehtesham.zahoor}@nu.edu.pk

Olivier Perrin and Ahmed Bouchami
Université de Lorraine, LORIA
BP 239 54506 Vandoeuvre-lès-Nancy Cedex, France
{olivier.perrin, ahmed.bouchami}@loria.fr

***Abstract*—Collaborative environments have put an enormous challenge to secure the information processing systems being used to manage them. Challenges to provide secure framework are amplified when it comes to the domain of flexible and distributed systems as the trust, temporal and performance related aspects need to be catered for. In this paper, we handle some security challenges among others the sub-mentioned ones by proposing a formal cloud-based authorization framework. We have considered trust to be a dynamic attribute to facilitate authorization decisions and have proposed models to handle different qualitative, quantitative and periodicity based temporal constraints. Further, we have presented an architecture for policies evaluation in the cloud.**

I. Introduction

The need to protect the information has always been there. When it comes to implementing security for an organization, the need for proper and detailed planning can be the road to success. Thus, we need a plan that specifies exactly how we would approach the challenge and this plan is termed as security Policy of an organization. There are different kind of security policies and in this paper we would focus on one important class of security policies, called the access control or authorization policies. An authorization process can be more specifically regarded as to determine, given some context, who can access what resources, under what conditions, and for what purpose. Authorization policies are high level descriptions of these access rules. In the last few years, the widespread adoption of the Cloud Computing has introduced new opportunities and the associated challenges. It has provided a model where the resources are pooled, shared and provided on demand over a network. They can be rapidly, elastically and even automatically provisioned for providing scalability and the users pay for what they use. From a client perspective, it has no idea about the exact location of the resources as they are dynamically provisioned and released. The widespread usage, benefits and offerings from different service providers are gaining momentum and businesses are seeking new opportunities to reduce hardware and management costs by offloading their capabilities to the cloud. However, with all the benefits associated with the cloud are some major challenges in terms of security and privacy. When the applications are moved to the cloud, we need new models for enforcing security.

The cloud based federated authentication has been an active area of research with approaches such as SAML providing SSO for federated environments, however the authorization capabilities, challenges and solutions are not thoroughly explored [1]. These challenges are amplified in a distributed and flexible environment. For cloud-based applications or resources, authorization should not only be performed based on the content, but also by the context. In addition, authorization assertions which refer to policies should be specified to be handled in a lightweight manner as well as encapsulating the most well known important policy requirements such as : role/group abstractions, obligation and fine-grained constraints, temporal aspect consideration, possibility to define permissions and prohibitions, rules and policies interactions, and handle both event and policies/rule conflict automatically. As our best knowledge, there is no existing policy model that fulfills all of these requirements together. In this paper, we propose the *CATT* (Cloud-based Authorization with Trust and Temporal aspects) framework. The proposed approach is based on event-calculus. CATT supports the sub-mentioned policy requirements with the computation and specification of authorization policies based on trust, temporal aspects and their combinations, in an efficient and flexible way. Specifically our contributions include:

A new formal authorization policies model: Our approach presents a new plan for defining access control policies based on attribute model (ABAC) and the Event-Calculus formal language. We believe that the logic-based languages are suited [2], [3] for the policy's definition, due to their openness for analysis and reasoning. Thus, policies becomes more accurate and expressive, with a high flexibility and non-ambiguous representation, and conflicts checking and goal refinement become easier. We have used the **abductive** reasoning type[1] [4], that consists in specifying the initial state(s) and the possible expected goal(s). In summary, with EC formalism, we were able to propose an expressive and generic policies framework, that takes into account the most important security requirement (from our point of view) that are: dynamic-context consideration, fluid role/group

[1]which is discussed by *Charniak and McDermott (1985,chap.8), Shanahan (1989;1997b, chap.17; 2000a), Denecker, Missiaen, and Bruynooghe (1992) and Hobbs, Stickel, Appelt and Martin (1993)*

COLLABORATECOM 2014, October 22-25, Miami, United States

DOI 10.4108/icst.collaboratecom.2014.257312

abstractions, obligations, fine-grained constraints, temporal authorization assertions aspect, deontic logic (permissions and prohibitions) [3], rules and policies interactions [5].

Generic pattern-based approach: Proposed EC models are organized in a generic pattern-based approach where individual simple patterns are enriched and combined to model complex requirements. Our approach opens new directions for authorization research as it can be extended to include design time verification [6], run time monitoring[7] and delegation[8].

II. Related work

Access control and authorization has been an active research area since decades and the focus of traditional research approaches has been the Role Based Access Control (RBAC) model and its variations [9]. Task based access control (TBAC) extends the traditional model by considering task based contextual information [10]. Even though RBAC is a well defined model, it suffers from *role explosion* as too many rules (may even surpass the number of users) may need to be managed. Some approaches have investigated the use and challenges for RBAC in a distributed environment [11], [12], [13], [14]. In contrast to RBAC models, the Attribute based access control (ABAC) model is based on the attributes [15]. The resources, subjects and environment have attributes and the policy rule is a boolean function on these attributes. ABAC subsumes RBAC and provides more flexibility and expressiveness than RBAC in term of rules definition as a role itself can be an attribute in an ABAC model. In a collaborative context, ABAC is thus the preferred model as it is more suitable for situations in which finer granularity and context-aware authorizations are required.

Several policy languages for network and security management was proposed in literature, and [2], [3] are good surveys[2]. Like our proposed policy model, PDL, REI, ASL, VALID, Webdamlog [16], OrBAC [17], CDL [18] are formal policy languages, but each of them suffers from some security requirements problems. For instance, PDL lacks hierarchical rules grouping and ordering [2], [3]. ASL, OrBAC and VALID are not event based, which makes them not adapted when considering a dynamic context. Furthermore, policies languages such as REI, PDL, Ponder need additional processing (meta-policies) for conflict handling.

Policies can also be defined in an XML form, like in XACML and EPAL [19]. XACML (eXtensible Access Control Markup Language) is a declarative, XML-based access control policy language for managing access to resources. It is based on ABAC model and the attributes are thus the most basic unit of a policy specified in XACML. As XACML is based on XML, a number of approaches to provide formal semantics of XACML using formal logic have been proposed [20], [21], [22]. A number of approaches have also been proposed to extend XACML for using it in collaborative and distributed environments. However, as XACML is not event-based, one of its major weakness is that it is not able to support the specification of policies under dynamic context. Furthermore, in collaborative and distributed environments, XACML suffers from other limitations. The first one is the problem of defining user level policies. In XACML, an enterprise can easily define a set of policies for every person within the enterprise, but it is more difficult for a user to define its own policy for managing access to its own data. In fact, the need for loosely coupled and user-centric authorization systems is increasing. XACML is verbose enough to lead to ambiguities that render difficult the possibility to analyze and validate a set of policies coming from various partners (both enterprises and users). This problem is even more important when it comes to give the ability to users to define access rules designed to be combined with the enterprise ones.

In this work, we have defined our policy model on top of the Event-calculus (EC) formal language [4]. First, we use Event Calculus for modeling policies patterns in the form of an Event-Condition-Action paradigm [5]. Therefore, we were able to model dynamic context changes and to consider them under our authorization model's scope. Second, EC allows us to represent (commonsense) knowledge and scenarios, and then to combine both to reason about these scenarios. Additionally, we were able to easily model authorization policies patterns with regard to different temporal aspect. Last, we can readily define authorizations as well as prohibitions assertions. Another asset of using EC in our work is the the ability to provide an automated reasoning thanks to the *DECReasoner*[3] tool for policy consistency checking and fluid conflict detection.

Regarding the concept of trust in a distributed and collaborative environment, it has been a highly active research direction in last few years [23], [24], [25]. Unfortunately, this interest from the research community has resulted in several different synonyms for the word trust. It can be either an indicator for multi-level security, a synonym for evidence and reputation, it can be used to signify the origin authentication and can have other meanings in some other contexts, as discussed in [26]. Most of the research about trust has been for the computation of trust and these approaches range from history based approaches to the ones based on statistical methods [27]. We believe that it should be possible to authenticate the origin of trust, being an important attribute. However, in this work we would not focus on the computation of trust but rather assume that it has been established, measured, associated or agreed upon. Our focus on trust-related aspects of this work is how to define efficient and expressive authorization policies that can incorporate

[2]Due to space limitation we avoid to give the reference for every policy language, interested people can find those lacking in the surveys.

[3]http://decreasoner.sourceforge.net/

trust, once it is already there. One important aspect of our proposal is that trust is decoupled from the authorization policy, which provides a greater modularity and a highly maintainable authorization framework. Further, the temporal constraints have been thoroughly investigated for the RBAC models and some seminal work is the GTRBAC model and its variants [28]. However the same is not true for distributed and collaborative ABAC models. The temporal constraints can be broadly categorized in three broad categories; quantitative, qualitative, and periodic constraints. The XACML policy language considers the limited qualitative constraints without catering for delays and durations.

III. Motivating Example

Our motivating example is based on one scenario coming from our industrial partners' and issued from the *OpenPaaS* project. We take the case study of a large enterprise, called *EnterpriseL*, comprising of many different departments such as sales, marketing and HR. As with any organization, it is prone to information *silos*. The different departments have developed their own culture and inter-department communication and collaboration is becoming difficult. The problem has also lead to difficulties in implementing security at *EnterpriseL* as heterogenous and localized authentication and authorization mechanisms exist not only across but even within departments.

The sheer size of the organization has an affect on the operational costs as well and the recent rise in the cloud computing offerings is the silver lining. The higher management has decided to shift to a (private) cloud to improve flexibility and reduce costs. It would also provide an opportunity to mitigate the problems caused by information silos and promote collaboration both within different departments of *EnterpriseL* but also with the other partner companies. The cloud based approach would allow to externalize the application level authorization policies to a central cloud based authorization provider (AzP), Figure 1. This would result in better management and ease in the specification of policies. However, the authorization challenges are still there. The enterprise needs fine-grained attribute based authorization policies that can be based on trust, temporal aspects and their combinations. These challenges remain and have amplified once they have moved to the cloud and traditional *RBAC* or *XACML* based approaches are proving limited as it comes at the price of performance, lack of flexibility and expressiveness, verbosity and lack of semantics.

We assume following information about some users. *James* is a trustworthy senior employee, belongs to *Auditing* department at EnterpriseL. *Bob* is a junior employee and has recently joined, belongs to *Auditing* department at *EnterpriseL*. *Natalie* is a senior manager at *EnterpriseL*. The first requirement relates to the performance aspects and is as below:

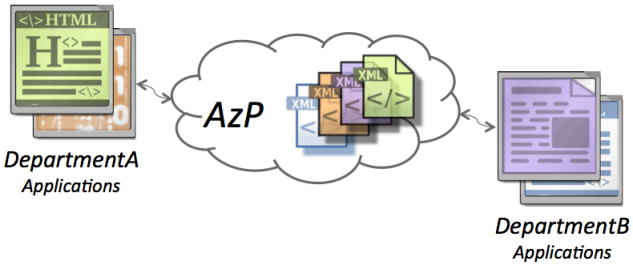

Figure 1. Motivating example - Cloud level authorization

> ***P1:*** *The approach should be efficient and provide timely decisions, probably by intelligent caching schemes and/or using an architecture to optimize bandwidth constraints and attributes availability.*

Then, the second class of authorization challenges we have focused in this work deal with the trust based authorization decisions. For the motivating example, we assume the case of internal auditing process and James and Bob from the Auditing department are working on it. We consider following requirements:

> ***TR1:*** *The auditing process involves access to a highly classified resource (AuditResource). James, being the senior trustworthy employee can access the resource and Bob is denied access.*
> ***TR2:*** *Once Bob completes the probation period, he can be given access without modifying policies.*
> ***TR3:*** *Natalie wants to access the AuditResource, she is denied access unless the report is published. The access is then given without modifying policies.*

Further we have some temporal constraints as well that relate to quantitative, qualitative and periodicity aspect. The temporal constraints can also be based on trust related aspects. The requirements for the motivating example include:

> ***TE1:*** *Permit decisions lasts for 5 minutes. Denied rules/policies cannot be reevaluated for next 2 minutes.*
> ***TE2:*** *At any given time only one user can be permitted to access a resource.*
> ***TE3:*** *Some events happen periodically, such as the user context or some policy is evaluated every 2 minutes to cater for dynamic environments.*

These requirements are difficult to handle using traditional approaches as they lack expressiveness and flexibility. The proposed approaches are either limited in expressiveness to model different types of constraints and more importantly their combinations, lack formal semantics to be verified and reasoned upon or have significant effect on the performance of the system. To best of our knowledge, their exists no approach that attempt to address these challenges in an unified and integrated way.

IV. Proposed Approach

The proposed approach for cloud based authorization provides both a formal policy specification language and an architecture to evaluate the policies. The specification language is based on Event-Calculus (EC) modeling formalism. The need for formal language is guided by the fact that we need to both model complex policies including temporal, security, cardinality and other constraints such as the reliability of the attributes being used to make the decision. A formal approach helps to elicit the decision, eliminating ambiguities. Further, policies can be defined at different levels and it may be required to combine them and find inconsistencies.

In terms of a cloud-based authorization approach, the need to specify the policies is indeed important but to evaluate them in an efficient way is equally important. We have thus proposed an architecture to evaluate the policies in the cloud. We have also targeted the challenges specific to a cloud setup such as the availability and reliability of attributes and the need to make the decision making process simpler without the cloud based authorization implementation to gather attributes for the policy evaluation.

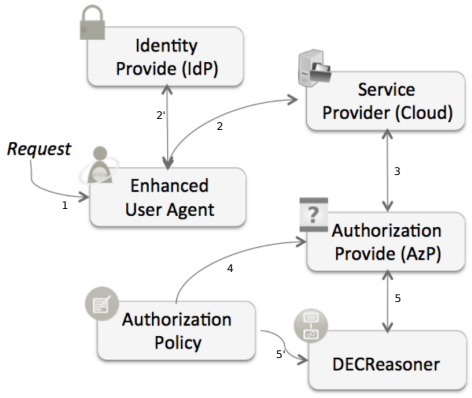

Figure 2. The proposed framework

A high level view of the proposed framework is shown in Figure 2. Once an authorization/access request to some resource is received by the cloud based service provider (SP), the user first needs to be authenticated by IdP to make the request. Once authenticated, the SP forwards the request along with the associated attributes to the Authorization Provider (AzP). The decision is taken by using the policy AzP maintains and reasoning about it using a constraints solver, called *DECReasoner* that can reason about event-calculus based models.

V. Policies Specification

In this section we will discuss the proposed approach for the specification of security polices. On a higher level, the proposed policy model has three basic constructs, the *Rules*, *Policies* and the *PolicySets*. The rules are at the core of the policy specification and each role has a *Target*, an *Effect*, a *Condition* and the associated *Trust-level* . The rule Effect may only be evaluated if the Trust-level is acceptable however, we will detail trust related aspects in next section. Individual rules can be combined to form a *Policy* and as each of the associated rules may evaluate (give decision) differently, a collection of rule combining algorithms are used (such as Permit Overrides for instance). The set of rule combining algorithms can be based on temporal and trust related aspects. Further, a *PolicySet* is a container for policies and has policy combining algorithms. The basic constructs to model the authorization policies have been intentionally chosen to have the same names as the ones used in XACML specification. This would allow us to easily transform an XACML based policy to the proposed model. However the differences exist as our approach is expressive to model temporal and trust based aspects.

The rest of this section is organized as follows. We first provide a brief background about event-calculus and then discuss the event-calculus based model for specifying different constructs without incorporating trust and temporal aspects, which would be detailed in the next section.

A. Event-calculus

Event-calculus (EC) is a logic programming formalism [29] for representing events and their effects. It comprises the following elements: $\mathcal{A}$ is the set of *events* (or actions), $\mathcal{F}$ is the set of fluents, $\mathcal{T}$ is the set of time points, and $\mathcal{X}$ is a set of objects related to the particular context. In EC, events are the core concept that triggers changes to the world. A fluent is anything whose value is subject to change over time. EC uses predicates to specify actions and their effects. Some basic event calculus predicates used for modeling the proposed framework are:

- $Initiates(e, f, t)$ - fluent f holds after timepoint t if event e happens at t.
- $Happens(e, t)$ specifies that event e happens at timepoint t.
- $HoldsAt(f, t)$ is true iff fluent f holds at timepoint t.

The event calculus models are presented using the discrete event calculus language [30] and we will only present the simplified models that represent the core aspects, intentionally leaving out the supporting axioms. All the variables (such as *rule*, *time*,...) are universally quantified and in case of existential quantification, it is represented with variable name within curly brackets, *{variablename}*. In order to simplify representation and to provide a generic approach that is not only limited to the motivating example, we would organize EC models in generic patterns. These patterns are discussed in a bottom-up manner where individual simple patterns are enriched and combined to model complex requirements. Further, this pattern based will allow us to implement tools to generate EC based models automatically.

B. Rules specification

The *rules* are at the core of the authorization policies and each rule has a *Target*, an *Effect* and the associated *Conditions*.

1) Rule target specification: The rule *Target* element specifies the attributes which are used to match if a particular rule applies to the input request. The proposed target specification approach is generic as it treats all the information needed as to be composed of name-value attributes. For instance, the *Resource*, the *Action*, the *Role/Name* of the user and other such information is considered as attributes having names and values. It thus allows for adding the new attributes for target specification if needed. We start our EC based modeling approach by discussing how the Target computation can be achieved using event-calculus. We take a simple requirement from the motivating example ***(TR1)*** about a rule called, *RuleJames*, which applies (has target) when the user *James* wants to have *write* access for an audit file, called *AuditResource*. We briefly discuss the pattern based approach in the model below:

Model 1 (Rule target specification)

```
;This part is a generic pattern.
sort rule       sort atname, atvalue
predicate AtHasValue (atname, atvalue)
event Match(rule), Mismatch(rule)       fluent RuleTargetHolds(rule)

Initiates (Match(rule), RuleTargetHolds(rule), time).
Terminates (Mismatch(rule), RuleTargetHolds(rule), time).
!HoldsAt(RuleTargetHolds(rule),0).
HoldsAt(RuleTargetHolds(rule),1) | !HoldsAt(RuleTargetHolds(rule),1).

;This part is specific to a particular rule being specified.
rule RuleJames
atname Subject, Object, Action, Context
Happens(Match(rule),time) & AtHasValue(Subject, atvalue1) &
AtHasValue(Object, atvalue2) & AtHasValue(Action, atvalue3) →
atvalue1 = James & atvalue2 = AuditResource & atvalue3 = Write.
;...similar axiom for Mismatch event
atvalue James, AuditResource, Write
AtHasValue(Subject/Object/Action,James/AuditResource/Write).
```

In the EC model shown above, there are two parts. One is the generic pattern that can be added to any EC model for target specification and the other is the instantiation of that model for any specific target specification and corresponding request. In the model, we first define some sorts, *rule*, *atname* and *atvalue*. We can consider sorts as *types* of which individual variables can be instantiated. The sort *rule* would be used to represent rules and *atname* and *atvalue* would be used to model attribute names and value respectively. We then define a predicate *AtHasValue* which specifies name-value pairs for attributes. Then, we define fluent, events and corresponding Initiates and Terminates axioms. The fluent *RuleTargetHolds* would be used as a goal and it does not hold at time-point 0. Then we define an event called *Match* and corresponding Initiates axiom that specifies that if the event happens at time *t*, that fluent will continue to hold from time point *t+1*. We additionally define another event, *Mismatch*, and corresponding *Terminates* axiom whose effect is opposite to the Initiates axiom. This completes the generic part for the model.

In the second part, we instantiate the generic model for a specific rule and request. We name the rule (by creating instance of sort rule) as *RuleJames*. Similarly we define a set of attributes, such as *Subject*, *Object* and others. Then we define a conditional axiom that the event *Match* can only happen if the attribute name value pairs match (and same for Mismtach event). Finally we define attribute values, that would be based on input request received and would change for each request. If we invoke the DECReasoner event-calculus reasoner on the event-calculus based specification, it returns a solution as shown below. The model shows that as the request (the one we intentionally specified) has the name value attributes that match with the rule target, this means the rule *Target* holds. In simpler words the current request applies to this rule. If we change any of the attributes like the *Action* is *Read* or *Subject* is *Bob*, the *DECReasoner* will provide a model which shows that the event mismatch would happen and the rule in question does not apply to it.

Solution 1 (Rule target result from DECReasoner)

```
18 variables and 51 clauses, relsat solver
model 1:
0 Happens(Match(RuleJames), 0).
1+RuleTargetHolds(RuleJames).
```

We can modify the Target specification model as shown above, by separating the generic part into individual files that can be included in this or any other model for target specification. The improved model is shown below. We separate sorts, and the core models in individual files and the model includes these files. In reference to the requirement identified for the motivating example, ***(TR1/TR3)***, we can similarly specify rule targets for for *Bob* and *Natalie*.We however, postpone trust related discussion to the next section.

Model 2 (Rule target specification - A pattern-based approach)

```
load includes/rules/sorts.e
load includes/rules/targets/core.e
load includes/inputs.e
;These generic patterns can be included in any model for target specification,
inputs.e models the request.

rule RuleJames
Happens(Match(rule),time) & AtHasValue(Subject, atvalue1) &
AtHasValue(Object, atvalue2) & AtHasValue(Action, atvalue3) →
atvalue1 = James & atvalue2 = AuditResource & atvalue3 = Write.
HoldsAt(RuleTargetHolds(rule),1) | !HoldsAt(RuleTargetHolds(rule),1).
```

2) Rule condition and effect: Once the target of the rule holds then the rule would be evaluated based on other information like the rule *Condition* and *Effect*. The rule Effect is to either *Permit* or *Deny* the rule, indeed once the *Target* or associated *Condition* hold. The rule effect can also be *Indeterminate* but to keep models simple, we intentionally do not consider the case. The rule Condition can be considered as a set of predicates that specify what

conditions we need to check for the rule.

The choice of expressive event-calculus based formalism allows us to define highly fine grained and extensive conditions that are not only based on the functional constraints but also on non functional constraints. The rule conditions can be based on constraints such as Separation of Duties (SoD), retention and integrity check and others. They can also include cardinality constraints, relations and temporal aspects. Further, the use of EC allows us to combine different non-functional constraints such as SoD constraints that are valid for some particular time interval. To this end, we have defined a fluent *RuleConditionHolds*, that specifies if the rule Condition holds or not.

3) Rule specification and evaluation: Once we have evaluated the rule *Target* and *Condition*, we can then evaluate the actual rule. A rule is considered to be permitted (or denied based on rule *Effect*) if the rule *Target* and rule *Condition* holds, the argument would however change with the incorporation of trust as we will discuss in next section. If the rule Target does not hold, the rule is considered *NotApplicable* and so is the case when the rule Target does hold but the associated *Condition* does not hold. In terms of EC model the basic idea is that once we have evaluated rule target, using *Model 3* and as represented by the fluent *RuleTargetHolds*, and evaluated Conditions, *RuleConditionHolds*, we can evaluate the rule based on their evaluation results. We present the EC model for rules specification below.

Model 3 (Rules specification - A pattern-based approach)

```
load includes/rules/core.e
;We use the complete Model 2, and add the rules pattern.

;We also update goal from the Model 2.
HoldsAt(RuleTargetHolds(rule),1) | !HoldsAt(RuleTargetHolds(rule),1).
HoldsAt(RuleIsPermitted(rule),2) | HoldsAt(RuleIsDenied(rule),2) | HoldsAt(RuleIsNotApplicable(rule),2).
```

In the model above, we use the complete Model 3 (with modifying the goal), and add another pattern, *includes/rules/core.e*, that would handle all the aspects related to rule computation in a generic way. The pattern is shown below. We first define some EC fluents which will be used to signify if the rule is applicable and if its effect is permit. Some other fluents will be used to signify that if the rule is permitted, denied or is not applicable. Then we specify an event called *ApproveRule* and corresponding initiates axiom. The next axiom restricts the event to only happen, if the rule target and condition are valid and its effect is to permit. We have similar events and axioms for *Deny* and *Not applicable* events which are not shown due to space limitations.

Model 4 (Rules specification pattern - /rules/core.e)

```
fluent RuleTarget/ConditionHolds(rule),RuleEffectIsPermit(rule)
fluent RuleIsPermitted/Denied/NotApplicable(rule)

event ApproveRule(rule)
Initiates(ApproveRule(rule), RuleIsPermitted(rule), time).
Happens(ApproveRule(rule),time)→HoldsAt(RuleTargetHolds(rule),time)&HoldsAt(RuleConditionHolds(rule),time)&HoldsAt(RuleEffectIsPermit(rule),time).
;Similar events/axioms for DenyRule and RuleDoesntApply.
;Not shown due to space limitations.

!HoldsAt(RuleIsPermitted/Denied/NotApplicable(rule),0).
```

If we invoke the *DECreasoner*, for the above event-calculus based specification, it returns a solution as shown in the model below. The solution shows that as the fluents that signify the rule condition, target and effect were all valid at time point 0, the event to permit the rule can happen at time point 0, resulting in the fluent *RuleIsPermitted* to hold at time point 1. However, if the rule condition fails or the effect of the rule is to deny the request then the fluent *RuleIsDenied* will hold at time point 1.

Solution 2 (Rules evaluation using DECReasoner)

```
0
RuleConditionHolds/EffectIsPermit/TargetHolds(RuleJames).
Happens(ApproveRule(RuleJames), 0).
1 +RuleIsPermitted(RuleJames).
```

4) Policy/PolicySets specification and evaluation: Individual rules that a user defines about a resource can be grouped to form a *Policy* and individual policies can be grouped in *PolicySets*. We would limit our discussion to *Policy* specification but the same approach applies to the specification and evaluation of *PolicySets*. We earlier discussed rules including, *RuleJames* and *RuleBob*. They can be combined in a policy, if needed. The EC model below shows how we can group individual rules in a Policy.

Model 5 (Policies specification - A pattern based approach)

```
load includes/rules/defined/RuleBob.e & RuleJames.e
load includes/policy/sorts.e & core.e & inputs.e
;These are policy/rule specific patterns. Some others are not shown.

policy ResourcePolicy
PolicyHasRule(ResourcePolicy, RuleJames).
PolicyHasRule(ResourcePolicy, RuleBob).
HoldsAt(PolicyIsPermitted(policy),3) | HoldsAt(PolicyIsDenied(policy),3) | HoldsAt(PolicyIsNotApplicable(policy),3).
```

The model above includes the patterns for different aspect related to rules specification and the actual rules as well, some patters such as *inputs.e* are not shown. The basic idea is that for each rule, once evaluated we will have evaluation results in the form of fluents. We can reason about those fluents to evaluate the policy result. The core pattern for policies is shown below. We have defined a policy that combines the two rules about the resource in question. One important aspect to consider here is that how different rules combine with each other. We can have a number of combining algorithms (such as permit-overrides, deny-

overrides and others). We can further have a PolicySet that is the combination of individual policies and corresponding policy combining algorithms. Space limitations restrict us to detail the models further.

Model 6 (Policies core pattern - includes/policy/core.e)

predicate PolicyHasRule(policy, rule)
fluent PolicyTargetHolds(policy),PolicyIsPermitted/Denied/NotApplicable(policy)
event ApprovePolicy(policy)
Initiates(ApprovePolicy(policy), PolicyIsPermitted(policy), time).

;Rule-combining algorithm. In this case permit-overrides others not shown.
Happens(ApprovePolicy(policy),time)→{rule}PolicyHasRule(policy,rule)&Holds At(PolicyTargetHolds(policy), time) & HoldsAt(RuleIsPermitted(rule), time).
!HoldsAt(PolicyIsPermitted/Denied/NotApplicable(ResourcePolicy),0).

VI. Trust based authorization policy

The event-calculus models shown in the previous section handle different aspects related to the policy specification and evaluation. We intentionally tried to present the simple models without incorporating the trust, temporal aspects or their combinations.

In this paper we would not focus on the computation of trust but rather assume that it has been established, measured, associated or agreed upon. Our focus on trust-related aspects of this work is how to define efficient and expressive authorization policies that can incorporate trust, once it is already there. We can define a *Trust-Level (TL)* to be computed based a 4 tuple $< S, O, A, C >$, where *S* denotes the trust level for the *Subject* (for instance the user making the authorization request), *O* signifies the critical nature of the *Object* (the resource in question for the authorization decision) and *A* signifies the nature of the action the subject wants to perform at the object. Finally the context C defines the environment in which the trust based decision is being taken. The environment may represent the conditions in which the decision is being taken and can be based on temporal and geographical attributes. For instance, during weekends or after working hours, or related to the location of the request. We will collectively term the trust for a *Subject*, the criticality measure for the *Object*, the severity level of the *Action* and the reliability of environment as the Trust Level (TL). We represent the TL with a number and its value ranges from 1 to 3 with the value 1 signifying the attribute to be not trusted, 2 signifying the attribute to be partially trusted while the value 3 means the attribute is fully trusted. The TL range for the *Context* can be from 0 to 2, where the 0 value signifying normal conditions.

When a user requests for a resource, the trust-level is computed based on the $< O, A, C >$ tuple and a value is computed. The critical nature of the resource and the action user wants to perform on the resource highlight the severity or criticality of the user action on resource. The criticality of the user action on specified resource is magnified based on the context in which the request has been made. For instance if the user wants to write on a critical resource, after working hours, weekend or a non-secure location, the criticality of user action on resource would indeed be amplified. Once a trust-level for the request has been computed based on the $< O, A, C >$ tuple, it is compared with the trust-level the subject has (TL_{usr}) and in case the subject has appropriate trust level, the decision can be computed.

More formally, let *AT* be the set that donates the set of attributes, $AT = \{at_1, at_2, at_3 ... at_n\}$ in a particular context C. In our case we consider the two attributes as *Object* and *Action*. Further, let *WT* be the set of weights assigned to the attributes, $WT = \{wt_1, wt_2, wt_3 ... wt_n\}$ where wt_1 is the weight assigned to attribute at_1. In our case, all attributes have the same weight, i.e 1. The trust decision can be modeled as below:

$$Decision = TL_{usr} \geq \left\lceil \frac{(\sum_{i=1}^{n} TL_{at_i}.wt_i) + TL_c}{\sum_{j=1}^{n} wt_j} \right\rceil \quad (1)$$

The formula above is intended to be flexible to handle a number of cases. For instance the weights of the attributes can be different to highlight one attribute, for instance giving preference to Object rather than the Action. However, the formula can be modified and even replace to handle any specific requirements. The use of a pattern based approach allows to easily achieve that. One important thing to note here is that the trust level does not overwrite other security policies but rather complement it. If the user request is trusted, he is still bound to the actually policies and rules for the resource. In relation to the proposed modeling approach and basic constructs defined in the last section for policy specification, we can modify the rule element. A rule has a Target, Condition, Effect and is evaluated based on the trust level computed for subject action on the resource in the context. If the trust level holds, the rule can be evaluate and otherwise, its effect can be deny.

We now present the event calculus model for trust based authorization decisions. We use the Rule specification model (Model 3) as shown earlier and and we need following modifications to the model.

1. *We need to updated the rules evaluation pattern (rules/core.e) to approve (deny) the rule, if the required trust level (does not) hold. We can simply update the axioms as one shown below:*

Happens(ApproveRule(rule), time) → HoldsAt(RuleTargetHolds(rule), time) & HoldsAt(RuleConditionHolds(rule), time) & HoldsAt(RuleEffectIsPermit(rule), time) & HoldsAt(TrustHolds(rule), time).

2. *We need to add the core patterns for trust,rules/trust/core.e and rules/trust/formula.e. They are explained below and contain core predicates and formula for computation trust approval.*

3. *We need to add a new sort trlevel: integer and specifyrange trlevel 1 3, to signify the trust level of attributes.*

4. *We need to update the input.e file to represent the trust related attributes with the input request, AttribTr-*

level(Subject/Object/Action, 1≤value≤3). For Context, 0≤value≤2.

We now present the core patterns for trust, as mentioned in the point 2 above. The pattern *rules/trust/core.e* contains all the events, fluent and corresponding axioms for trust specification and evaluation, as shown below:

Model 7 (Trust-based authorization - rules/trust/core.e)

```
predicate AttribTrlevel (attribute, trlevel)
predicate RequestTrlevel(trlevel)
event TEvalSuccess(rule) event TEvalFailure(rule)
fluent TrustHolds(rule)

Initiates (TEvalSuccess(rule), TrustHolds(rule), time).
Terminates (TEvalFailure(rule), TrustHolds(rule), time).
!HoldsAt(TrustHolds(rule),0).
```

In the model above, we first defined some predicates that are used to associate the attributes with their trust level. Then we define events, fluent and associated Initiates and Terminates axioms, which signify if the trust level for the rule holds or not. We need to also include the pattern *rules/trust/formula.e* that contains the actual formula used for calculating if the trust holds or not, as shown in Equation 1. The event calculus model is shown below:

Model 8 (Trust-based authorization - rules/trust/formula.e)

```
AttribTrlevel(Object, trlevel1) & AttribTrlevel(Action, trlevel2) & Attrib
Trlevel(Context, trlevel3) → RequestTrlevel((trlevel1+trlevel2+trlevel3)/3).

Happens(TEvalSuccess(rule),time) & AttribTrlevel(Subject, trlevel1) &
RequestTrlevel(trlevel2) → trlevel1>=trlevel2.
```

The points 3 and 4 require us to update the input.e file to represent the trust related attributes with the input request. This file contains the actual trust level for attributes, as known to the system or as computed based on request. These values can be dynamic and can be either fetched from database or computed on the fly, scope of the paper restricts us to detail this aspect further.

In relation to the motivating example, the complete rule specification (with trust and other aspects) for the rule named *RuleBob* is shown below.

Model 9 (Trust-based authorization - RuleBob.e)

```
load includes/rules/sorts.e & rules/core.e & trust/core.e & /trust/formula.e &
targets/core.e & load includes/inputs.e

rule RuleBob
Happens(Match(rule),time) & AtHasValue(Subject, atvalue1) &
AtHasValue(Object, atvalue2) & AtHasValue(Action, atvalue3) →
atvalue1 = James & atvalue2 = AuditResource & atvalue3 = Write.
;Similar axiom for Mismatch event.

HoldsAt(RuleEffectIsPermit(RuleBob),0).
load includes/rules/rulesgoal.e
```

The contents of the authorization request message, as modeled in the input.e are as follows.

Model 10 (Trust-based authorization - inputs.e)

```
atname Subject, Object, Action, Context
atvalue Bob, AuditResource, Write
AtHasValue(Subject,Bob)/(Object,AuditResource)/(Action,Write).
AttribTrlevel(Subject,2).
AttribTrlevel(Object,3). AttribTrlevel(Action,2).AttribTrlevel(Context,0).
```

If we invoke the DECReasoner for the above rule, it returns a solution as shown below. The solution shows that the request message matches (target holds) the rule but the trust level of the use is not sufficient according to Equation 1, (2<3), the decision is to deny access. Then for the motivating example, ***TR2*** requires to allow access once the probation period is over for the Bob. Once that happens, the trust level of Bob would increase and will automatically allowed access, according to Equation 1, (3≥3). Same is the case for the requirement, ***TR3*** , once the *AuditResource* is finalised its critical nature would decrease and *Natalie* would be provided access.

Solution 3 (Trust-based authorization - RuleBob.e)

```
0: Happens(Match(RuleBob), 0).
1: +RuleTargetHolds(RuleBob). Happens(TEvalFailure(RuleBob), 1).
2: Happens(DenyRule(RuleBob), 2).
3: +RuleIsDenied(RuleBob).
```

VII. Temporal aspects

Time is intrinsic in every aspect of human life and has to be catered to model any related aspect. The need to thus model the temporal aspects (or constraints as we will term them) in modeling the information security perspectives is evident. Further, to model the temporal constraints we need an approach that can cater for time-point, intervals and their combinations to handle the periodic events. The choice of event-calculus as a modeling formalism is evident as EC integrates an explicit time structure and that is not the case with similar formal approaches such as situation-calculus. In this section we would first the classes of temporal constraints and their EC based models. Then we would discuss how the policy and rule combing algorithms can be based on the temporal aspects.

A. Constraints

The temporal constraints can be broadly categorized in three categories; quantitative, qualitative, and periodic constraints. The quantitative constraints consider the time in a metric sense and basic operations such as additions, subtractions, comparisons and others on the time variables can be applied. For instance, in terms of authorization constraints, consider the rule that a person can access a resource on some specific date between 9AM to 5PM (or between dates). The quantitative constraints can also include the duration constraints, such as once a request is granted, it lasts for next 5 hours and also the delays between different events can be specified. The qualitative constraints relate to the particular relative position of different entities. For instance, in terms

of authorization constraints, consider the requirement that the a set of rules should only be evaluated at the same time, or one before/after another, or their evaluation should finish at the same time and others. Finally the periodic constraints can be both qualitative and quantitative and they are based on periodicity/repetition of events.

For the quantitative constraints representation, we consider the temporal requirement ***TE1***, for the motivating example. Once a policy is permitted, the decision lasts for 5 minutes and once it is denied it cannot be reevaluated in next 2 minutes. The EC axioms below handle these requirements:

Happens(ApproveRule(rule),time1) & (Happens(ApproveRule(rule),time2) | Happens(DenyRule(rule), time2) | Happens(RuleDoesntApply(rule), time2)) → time2-time1=5
Happens(DenyRule(rule),time1) & (Happens(ApproveRule(rule),time2) | Happens(DenyRule(rule), time2) | Happens(RuleDoesntApply(rule), time2)) → time2-time1>=2

For the qualitative temporal constraints, two broad approaches are to specify the constraints either based on the intervals or base them on the time points. Interval based temporal constraints can be based upon the Allen's Interval Algebra, which is a calculus for temporal reasoning and was introduced by James F. Allen in 1983. The calculus defines possible relations between time intervals and provides a composition table that can be used as a basis for reasoning about temporal descriptions of events[4]. All the base relations for the Allen's interval algebra can be mapped and applied to the proposed approach, a detailed discussion can be found in [31]. The qualitative constraints can also be based on time-points as proposed in the points algebra [32], which is computationally less expensive as compared to Allen's Interval algebra as the base relations are only $<, =$ and $>$. As the notion of time-points are intrinsic in EC models and they can be compared using a set of operators, it can easily model the points algebra as well but due to space limitations, we would not detail them.

For the specification of qualitative temporal constraints, we take the requirement **TE2** from the motivating example that at any given time-point or time-interval, only one access should be permitted. The following generic EC axioms can handle this requirement.

Happens(ApproveRule(rule1),time) & rule1 != rule2 → !Happens(ApproveRule(rule2),time).
Happens(ApproveRule(rule1),time1) & Happens(ApproveRule(rule1),time2) & time1 != time2 & time3 > time1 & time3 < time2 → !Happens(ApproveRule(rule2),time3).

Further, we consider another class of temporal constraints that relate to periodic nature of events. These periodic temporal constraints can be based on both quantitative and qualitative aspects. Again as the EC involves *events* that *happen* at *time-points*, it provides an expressive approach to model periodic constraints. The EC axioms below handle the requirement **TE3** from the motivating example that a policy should be evaluated every 2 minutes, *InvocationEvent* can be some or all of *Approve/Deny/DoesNotApply* events.

Happens(InvocationEvent(SomePolicy), time1) & time2-time1 = 2 → Happens(InvocationEvent(SomePolicy), time2).

B. Conflict management

As our approach is event-based, events conflicts can occur. [5] and [33] discuss this issue. In our proposal, thanks to the Event-Calculus expressiveness, we are able to model an ordering and we configure the events occurrences with a simple predicate. Thus, we can avoid conflicting events. The second aspect for conflict management is to deal with conflicts that can arise during the rules and policies combination. The rule/policy combination algorithms specify what to do in case of conflicts thanks to basic algorithms such as Permit and Deny-Overrides. Such strategies exists in other policies languages like XACML, ASL or REI. We take the concept one step further, as a policy groups rules (and so as a PolicySet groups policies), it is indeed possible to specify the relational temporal constraints about rules/policies as a rule combining algorithm.

C. Evaluation

The proposed evaluation architecture is well suited to the cloud based AzP as it does not have to fetch the attributes for decision making rather the attributes are provided by the user agent. Few important points need to be addressed here. First the AzP and IdP are independent as the IdP can be external or AzP may need to work with more than one IdPs.

Two test cases have been evaluated based on the EC models for the motivating example, with temporal and trust based constraints resulting in a complex model. First we increase in the number of Rules within a *Policy* and policies within *PolicySet* (up to a combination of 200 rules). Then, we further complicated the models we combined different number of constructs. The solution computation is very efficient as even with the most complicated models, the time taken is less than 2 seconds on a PC machine. The EC-SAT encoding process also performs reasonably well and can be further improved by using incremental encoding or by further improving *DECReasoner* code.

VIII. Conclusion

In this paper we propose a formal Event-Calculus (EC) based cloud-based authorization framework, called *CATT*, which supports the efficient computation and specification of authorization policies based on trust, temporal aspects and their combinations. In contrast to traditional XML based policy languages, our approach is formal which provides a precise, expressive, flexible and non-ambiguous representation, and also allows for reasoning about authorization policies (e.g. to find inconsistencies). We have organized the EC models in a generic pattern-based approach where

[4]http://en.wikipedia.org/wiki/Allen's_Interval_Algebra

individual simple patterns are enriched and combined to model complex requirements. The proposed EC models can be easily modified and extended to handle other aspects.

REFERENCES

[1] R. Poortinga-van Wijnen, B. Hulsebosch, J. Reitsma, and M. Wegdam, "Federated authorisation and group management in e-science."

[2] W. Han and C. Lei, "A survey on policy languages in network and security management," *Computer Networks*, vol. 56, no. 1, pp. 477–489, 2012.

[3] N. Damianou, A. Bandara, M. Sloman, and E. Lupu, "A survey of policy specification approaches," *Department of Computing, Imperial College of Science Technology and Medicine, London*, vol. 4, pp. 1–37, 2002.

[4] E. T. Mueller, *Commonsense reasoning*. Morgan Kaufmann, 2010.

[5] J. Lobo, R. Bhatia, and S. A. Naqvi, "A policy description language," in *AAAI/IAAI*, 1999, pp. 291–298.

[6] E. Zahoor, K. Munir, O. Perrin, and C. Godart, "A bounded model checking approach for the verification of web services composition," *Int. J. Web Service Res.*, vol. 10, no. 4, pp. 62–81, 2013.

[7] E. Zahoor, O. Perrin, and C. Godart, "An event-based reasoning approach to web services monitoring," in *ICWS*, 2011.

[8] K. Gaaloul, E. Zahoor, F. Charoy, and C. Godart, "Dynamic authorisation policies for event-based task delegation," in *CAiSE*, 2010.

[9] J. S. Park, R. S. Sandhu, and G.-J. Ahn, "Role-based access control on the web," *ACM Trans. Inf. Syst. Secur.*, vol. 4, no. 1, pp. 37–71, 2001.

[10] R. K. Thomas and R. S. Sandhu, "Task-based authorization controls (tbac): A family of models for active and enterprise-oriented autorization management," in *DBSec*, 1997.

[11] E. Freudenthal, T. Pesin, L. Port, E. Keenan, and V. Karamcheti, "drbac: Distributed role-based access control for dynamic coalition environments," in *ICDCS*, 2002, pp. 411–420.

[12] T. Wu, X. Pei, Y. Lu, C. Chen, and L. Gao, "A distributed collaborative product design environment based on semantic norm model and role-based access control," *J. Network and Computer Applications*, vol. 36, no. 6, pp. 1431–1440, 2013.

[13] C. Ruan and V. Varadharajan, "Dynamic delegation framework for role based access control in distributed data management systems," *Distributed and Parallel Databases*, vol. 32, no. 2, pp. 245–269, 2014.

[14] H. K. Lee and H. Luedemann, "lightweight decentralized authorization model for inter-domain collaborations," in *SWS*, 2007, pp. 83–89.

[15] V. C. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone, "Guide to attribute based access control (abac) definition and considerations," *NIST Special Publication*, vol. 800, p. 162, 2014.

[16] S. Abiteboul, E. Antoine, G. Miklau, J. Stoyanovich, and V. Z. Moffitt, "Introducing access control in webdamlog," *CoRR*, vol. abs/1307.8269, 2013.

[17] F. Cuppens and A. Miège, "Modelling contexts in the or-bac model," in *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*. IEEE, 2003, pp. 416–425.

[18] R. Thomas and S. Tsang, "Cdl: A language for specifying high-level cross-domain security policies," in *Military Communications Conference, 2008. MILCOM 2008. IEEE*. IEEE, 2008, pp. 1–7.

[19] P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter, "Enterprise privacy authorization language (epal 1.2)," *Submission to W3C*, 2003.

[20] J. Bryans, "Reasoning about xacml policies using csp," in *SWS*, 2005, pp. 28–35.

[21] T. N. Nguyen, K. T. L. Thi, A. T. Dang, H. D. S. Van, and T. K. Dang, "Towards a flexible framework to support a generalized extension of xacml for spatio-temporal rbac model with reasoning ability," in *ICCSA (5)*, 2013.

[22] V. Kolovski, J. A. Hendler, and B. Parsia, "Analyzing web access control policies," in *WWW*, 2007, pp. 677–686.

[23] C. Qu and R. Buyya, "A cloud trust evaluation system using hierarchical fuzzy inference system for service selection," in *AINA*, 2014, pp. 850–857.

[24] A. Öksüz, "Turning dark into white clouds - a framework on trust building in cloud providers via websites," in *AMCIS*, 2014.

[25] F. Moyano, K. Beckers, and M. C. F. Gago, "Trust-aware decision-making methodology for cloud sourcing," in *CAiSE*, 2014, pp. 136–149.

[26] D. Gollmann, "From access control to trust management, and back - a petition," in *IFIPTM*, 2011, pp. 1–8.

[27] M. Firdhous, S. Hassan, and O. Ghazali, "Statistically enhanced multi-dimensional trust computing mechanism for cloud computing," *IJMCMC*, vol. 5, no. 2, pp. 1–17, 2013.

[28] J. Joshi, E. Bertino, and A. Ghafoor, "Temporal hierarchies and inheritance semantics for gtrbac," in *SACMAT*, 2002.

[29] R. A. Kowalski and M. J. Sergot, "A logic-based calculus of events," *New Generation Comput.*, vol. 4, no. 1, 1986.

[30] E. T. Mueller, *Commonsense Reasoning*. CA, USA: Morgan Kaufmann Publishers Inc., 2006.

[31] E. Zahoor, "Gouvernance de service: aspects sécurité et données," Ph.D. dissertation, Université Nancy II, 2011.

[32] M. B. Vilain and H. A. Kautz, "Constraint propagation algorithms for temporal reasoning," in *AAAI*, 1986.

[33] J. Chomicki, J. Lobo, and S. A. Naqvi, "A logic programming approach to conflict resolution in policy management," in *KR 2000, Principles of Knowledge Representation and Reasoning Proceedings of the Seventh International Conference, Breckenridge, Colorado, USA, April 11-15, 2000.*, 2000, pp. 121–132.

[34] L. Kagal, T. Finin, and A. Joshi, "A policy language for a pervasive computing environment," in *Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003. IEEE 4th International Workshop on*. IEEE, 2003, pp. 63–74.