Trust-Based Formal Delegation Framework for Enterprise Social Networks

Ahmed Bouchami, Olivier Perrin, Ehtesham Zahoor

Trust-based formal delegation framework for Enterprise Social Networks

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

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

***Abstract*—Collaborative environments raise major challenges to secure them. These challenges increase when it comes to the domain of Enterprise-Social-Networks (ESNs) as ESNs aim to incorporate the social technologies in an organization setup while asserting greater control of information security. In this context, the security challenges have taken a new shape as an ESN may not be limited to the boundaries of a single organization and users from different organizations can collaborate in a common federated environment. In this paper, we address the problem of the authorization's delegation in federated collaborative environments like ESNs. In contrast to traditional XML based languages, such as XACML, our approach is based on event-calculus, a temporal logic programming formalism. Further, the traditional approaches are either user-centric or organization-centric. However, the domain of ESN requires to bridge the gap between them and the proposed framework deals with this challenge. In order to enhance the delegation scheme, we have proposed a behavior monitoring mechanism, that permits to assess principals' trust level within the federated collaborative environment. We evaluate our trust computing approach based on simulated principals' behaviors and discuss the obtained results.**

I. Introduction

Since the last decade, *enterprises social networking* (*ESN*) emerged. *ESN*s offer for professional (enterprises or more generally organizations) a new way to easily exchange information, avoiding some abusive administrative procedures which often could take too much time and effort. However, when it comes to professional exchanges, the information could be sensitive and highly confidential. Thus, such exchanges need to be supervised and managed with great attention. This could be managed by an **access control** mechanism [1] on the top of the social environment [2], [3], [4].

Enterprise social network like Yammer[1], Tibco Software Tibbr[2], or eXo[3] are fundamentally collaborative environments in which several users belonging to different organizations could exchange data under a dynamic, trusted and secure framework. Taking this requirement into account, we can consider such environment as a *federation* [5]. From a security point of view, this leads us to define an *enterprise-social-network* as *"a multitude of communities, in each of them, involved active entities (users, organizations, web-services,...) trust each other concerning identity attributes, and, are subjected to an access control rules that govern the access to the shared resources"*. The access control rules concern a user, a resource and an action (S/O/A), and in such environment, these rules are often defined by the owner of the resources. We use the term *subject* to refer to users, principals or automated manager components [6].

As a community is a dynamic environment, resources and subjects tend to change frequently (*i.e.*, add-in and/or suspend-from the community) and could be not permanently available. Therefore, permissions should be frequently updated to be adapted to the real-time changes concerning community's subjects and resources availability. Such an issue is manageable using the attribute-based access control ABAC [7]. However, the challenge is when a subject, let's say *Bob*, should quit *temporally* the community, while his tasks must be performed within the community, because they are related to an existing process. In such case, *Bob* would choose another member, let's say *Alice* to replace him for performing his task(s) during his temporal leave. However, *Bob*'s permissions should be redefined to allow *Alice* fulfilling *Bob*'s tasks. This also could be handled if *Bob* defines a permission(s) for *Alice* to access his resource(s). However, in this case, *Bob* does not remain the only main owner of the resource. Further, even if *Bob* is available to complete his task(s), *Alice* remains able to use *Bob*'s resource(s). To deal with this issue, a *delegation* [8], [9], [10], [11] model could be an efficient solution. In the literature, a delegation is defined as a temporary transfer of access rights [6]. So, rules should be strongly time-based for defining the validity of the granted privilege(s). Yet, it is not trivial to manage the temporal constraints for subjects who delegate their access rights, especially when they have a lot of permissions to delegate.

To deal with the above mentioned issues, we have defined a delegation process as a transfer inside a community of one or more permissions from a subject to another belonging to the same community. The delegated permissions are temporary and they are only valid in the case of *delegator*'s[4] unavailability in the community. Therefore, the first challenge we address in this work is to define a *delegation framework* that ensures keeping subjects' hierarchy regarding to resources ownership (*i.e.*, privileges).

A classical problem for delegation in real life deals with trust. People normally must *trust* [12], [13] persons they delegate. In addition, the delagatee[5] should respect the *delegator*

[1]https://www.yammer.com/
[2]http://www.tibbr.com
[3]http://www.exoplatform.com/company/en/resource-center

[4]Who delegates.
[5]Who gets the delegation.

access rights. That's mean that the *delegatee* is supposed to only access the *delegator*'s permitted resources within the community. However, the *delegatee* behavior is unpredictable and he/she may make some malicious attempts to access unauthorized resources at the level of organizations' servers that host them (Fig 1). Thus, we clearly identify the relation between subject's trust-level and rights delegation in our ESN context.

To deal with this challenge, we consider subjects' trust-level within a community as a very important criteria to take into account for a delegation process enforcement within a community. In addition, we aim to ensure *organization's autonomy* to control the access to the delegated resources. This can be done by means of *trust-level* based constraints. Therefore, we need to efficiently monitor subjects' behavior within each collaborative environment, and accordingly increase and/or decrease their respective trust-levels over collaborative sessions within the *ESN*'s communities.

To summarise, we propose in this work a *generic trust assessment mechanism* on which we build our *formal delegation framework* for professional collaborative platforms. The formal delegation framework is based on Event-Calculus [14] and is designed to be combined with the access-control policy model we proposed in [15]. The generic trust assessment mechanism could be adopted at the level of any collaborative environment.

The rest of this paper is structured as follows. In section II, we present a global overview of our architecture to better illustrate the context of this work. We follow this by a motivating example to underline our objectives. Then, we discuss some related work in section III. Further, we present our formal delegation framework in section IV. In sectionV, the trust monitoring mechanism is presented and detailed. Then, simulation results are discussed. Finally, we conclude the paper and discuss the outline of our future work.

II. Architectural Vision and Case Study

A. Architecture

Our vision of the enterprise social networking is mainly based on the communities concept. A community is a logical group of collaborative subjects sharing collaborative resources. A community is created by a subject (having the required rights) which we call *creator*.

In a professional context, we assume that subjects' resources are stored within their organizations. In our architecture, the access-control is community centralized. That means that it is enforced through a common module for all organizations. Thus, when a subject shares a resource with another subject, an access-control rule will be defined at the level of the community's access-control database. Therefore, the access control decision will be enforced by the community's access control module. The access will concretely occur at the level of the organization's server that hosts (*i.e.*, owns) the resource. In such context, organizations would keep their autonomy to control the access on the shared resources they host. To deal with this vision, we propose an architecture (Fig.1) in which organizations remain fully-independent by means of access constraints added to the initial access control rule (set by a user). This constraint (set by the organization) consists in a

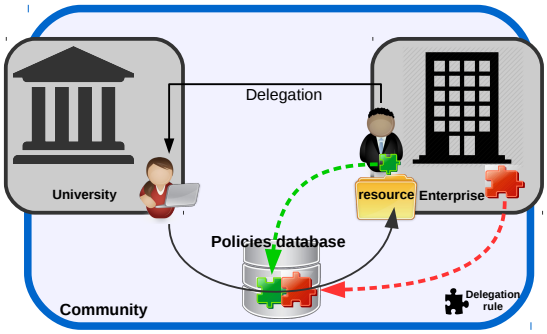

Fig. 1: Community access control global view.

threshold a requester must satisfy to been granted the access to the desired resource.

As we use the some subjects' attributes *trust* for the access-control enforcement, we have used an *Attribute Based Access Control* [7] *ABAC* model. We believe that *ABAC* is the most suited model for our designed delegation framework, because it is flexible and ideal for many distributed or rapidly changing environments [7]. As depicted in Fig 2, our delegation-access-control framework is composed of five main components namely:

1) **Community Enforcement Service (CES)**: the central unit of the enforcement of the access control policies. It also serves as a bridge for the interaction between the other modules.
2) **Community Information Store (CIS)**: the community's information database, where attributes about resources, subjects and organizations are stored.
3) **Community Administration Store (CAdS)**: the central community's delegation-access-control rules database.
4) **Community Decision Service (CDS)**: the module responsible for looking for access rules from the *CAdS*, generating the event-calculus pattern and enforcing the access-control decisions based on the DEC-reasoner.
5) **Community Trust Service (CTS)**: this service gets the subject session's information from the *CES* and the subject's historical information from the *CIS*, then, computes the *delegatee* trust-level, and finally update it in the *CIS*.

B. Motivating Example

For the motivating example, we consider the case of a charitable organization[6]. To avoid terms ambiguity, we will call it *association*. The organization creator – let's say *James* – wants to get the benefits of social networking to facilitate the collaboration between the association members. Therefore, *James* creates a community within the *ESN* and invites his partners to join the community. *Jessy*, the association Blog manager, is responsible to manage the association's directory. *Alice*, a journalist, writes articles about the association events. *Oscar*, a financial officer, manages the financial aspects of the

[6]http://en.wikipedia.org/wiki/Charitable_organization

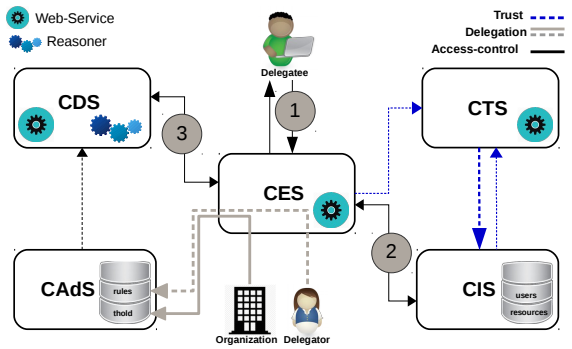

Fig. 2: Delegation access control architecture.

association. Then, *Dave* is an officer in the bank in which the association's financial funds are placed. Let us consider the following scenarios:

- **CS1**: *Jessy* delegates her rights to *Alice* for updating the association's calendar of solidarity. *Alice* behavior within the community is normal and more-or-less stable.
- **CS2**: *Dave* delegates his rights to *Oscar* to be able to manage the association's financial documents stored at the level of *Dave*'s bank. *Oscar*'s experience within the community reflects a unreliable behavior.

III. State of art

There has been a lot of research work to address the delegation issues. The most famous one is RBDM0 (Role-Based Delegation Model 0), and its variants RBDM1 and RDM2000 [16], [17], [18], [19]. These proposals are role-based models for simple, hierarchical role delegation and multi-step delegation respectively. The limit of those models is that the minimum delegation unit is the role, and they do not permit to delegate a subset of the permissions from a given role. In our context, such partial delegation is very important as we take into account the members' attributes heterogeneity. More precisely, within a collaborative environment, the same role (status) in different organizations does not allow the same privileges. In [20], the authors proposed the work entitled *a flexible role-based delegation model with dynamic delegation role structure*. However, this model does not allow ad-hoc user-to-user delegation as it is based on mapping between roles, capabilities and permissions.

XACML (eXtensible Access Control Markup Language) is a declarative, XML-based access control policy language for managing access to resources. A number of approaches have been proposed that build upon XACML for its usage in collaborative and distributed environments. In [21], the authors have developed a formal policy language BelLog that can express both delegation and composition operators. In [22] was proposed an XACML-based policy administration control and delegation model with the use of the *Delegent* server [23]. This model is capable to express chains of delegation and constraints on delegation using XACML. However, basic access delegations are governed by an administrative authorizations to avoid conflicting delegations. Whilst, with our formal delegation model, conflicts between delegation rules is detected automatically during the reasoning process.

[13], [24], [11] are formal delegation models. [24] is an event-calculus based model used for task delegation in workflow system. Stephen S. Yau et al propose in [11] a formal trust based delegation framework, based on AS^3 logic. However, they do not compute the trust based on the user's history, they consider a principal as trusted if its identity is authenticated and the access control permission is granted to it. The model proposed in [13] allow conditional delegation. [13] is fully distributed, thus, every organization would have its own reasoning mechanism to keep its autonomy to control its resources, otherwise they passively delegate this task to an external agent. With our approach the reasoning process is centralized within a federation and keeps organizations' autonomy by means of setting additional access conditions with a simple attribute sending.

Another important field in the trust computing research area deals with the probabilistic models. Indeed, many works use the beta model [25], [26], [27], [28], [29]. These models use a parameter Θ to estimate the probability if a principal will have a good or bad $(1 - \Theta)$ behavior regarding an expected behavior. [30] addresses the challenge by approximating the unknown parameter Θ based on the user's interaction history. We actually do the same, estimating the future user's behavior based on its historical behavior. However, in our approach, we do not need a probabilistic parameter, we instead aim to determine the severity with which the principal will be penalized based on the completed actions.

Our proposed framework is semi-distributed, which means that resources stay hosted and controlled at the level of their organizations and principals identities are managed by a central entity under a federation to avoid the problem of identity attributes heterogeneity. In contrast to traditional XML based access control policy specification languages, our approach is formal. This provides a precise, expressive, flexible and non-ambiguous representation, and also allows for reasoning about delegation policies (e.g. to find inconsistencies or hard constraints). We also take advantage from statistical models to efficiently monitor users' behavior within the collaborative environment and adjust the trust level.

IV. Delegation and access-control

A. Event-Calculus background

Event-calculus (*EC*) is a logic programming formalism for representing events and their effects on environment's attributes. 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. A detailed discussion about event-calculus can be found in [31]. Some basic event calculus predicates used for modelling the proposed framework are:

- $Initiates(e, f, t)$: fluent f holds after time-point t, if event e happens at t.
- $Happens(e, t)$: event e happens at time-point t.
- $HoldsAt(f, t)$: is true if fluent f holds at time-point t.

The event calculus models are presented using the discrete event calculus language [32]. We have used the Abductive reasoning type[7], that consists in specifying for up-stream the initial state(s) and the possible expected goal(s). This blends well with our vision of delegation policies model, as the system's behavior in a given state is susceptible to change due to the occurrence of some controlled events.

B. Formal delegation framework

The core component of our formal delegation framework is the delegation-rule ($Model2$). When it receives an incoming authorization request, the *CDS* tries to find the target ($Model3$) of the request among a set of delegation-rules. The set of the delegation-rules is called a *Policy* ($Model1$), which groups the requester's delegation rules issued from the *CAdS*. For each rule, the requester must satisfy the additional *trust* condition ($Model4$). In addition, a delegation-rule is valid if and only if the *delegator* is not available ($Model5$).

When a subject requests a resource as a *delegatee*, the system looks for all its delegation *rules* and groups them into a common set, the **"Policy"**. Then, the *Policy* will be evaluated based on the policy-model depicted in ($Model1$).

Model 1 (Delegation Policy model)

sort p, r
fluent $[PolicyPermitted/PolicyDenied/PolicyNotApplicable](p)$
event $[ApprovePolicy/DisApprovePolicy/RejectPolicy](p)$
$predicate\ PolicyHasRules(p, r)$

$\forall p, t:\ Initiates(ApprovePolicy(p), PolicyPermitted(p), t).$
$\forall p, t:\ Initiates(DisApprovePolicy(p), PolicyDenied(p), t).$
$\forall p, t:\ Initiates(RejectPolicy(p), PolicyNotApplicable(p), t).$

#Combination strategy: Permit-takes-precedence:
$\forall p, r, t:\ Happens$*(ApprovePolicy*$(p), t) \rightarrow$
$\exists r : PolicyHasRules(p, r)\ \&HoldsAt(RulePermitted(r), t).$

$\forall p, r, t:\ Happens$*(DisApprovePolicy*$(p), t)\&PolicyHasRules(p, r) \rightarrow$
$HoldsAt(RuleDenied(r), t).$

$\forall p, r, t\ :\ Happens$*(RejectPolicy*$(p), t)\&PolicyHasRules(p, r)\ \rightarrow$
$HoldsAt(RuleNotApplicable(r), t).$

In this policy model, we have defined the policy sort p, the rule sort r, and some *fluents* and *events*. For each *event*, we have defined an *action precondition axioms*. The latter is a requirement that must be satisfied for the occurrence of an *event*. An *event* whose action precondition is not satisfied cannot occur. The *Policy* model is based on the *rule* combination strategy *permit-takes-precedence*. This means that, for a given requester, if the enforcement system (*CES*) finds at least one permitted delegation rule, then, the subject will be authorized. At the end of the model, we initialize the *fluents* as not held, which means that they will still not held until the raise of their corresponding events which are subjected to conditions involving *rules*. Therefore, we would discuss the *rules* verification in the model depicted below : *Model 2: Delegation Rule model.*

Model 2 (Delegation Rule model)

sort r, s
fluent $[RulePermitted/RuleDenied/RuleNotApplicable](r)$
event $[ApproveRule/DisApproveRule/RejectRule](r)$

$\forall r, t:\ Initiates(ApproveRule(r), RulePermitted(r), t).$
$\forall r, t:\ Initiates(DisApproveRule(r), RuleDenied(r), t).$
$\forall r, t:\ Initiates(RejectRule(r), RuleNotApplicable(r), t).$

$\forall r, t, s:\ Happens$ *(ApproveRule*$(r), t) \rightarrow HoldsAt$ *(TargetHolds*$(r), t)$
$\&\ HoldsAt$ *(TrustHolds*$(s), t)\ \&\ HoldsAt$ *(RuleEffectIsPermit*$(r), t).$

$\forall r, t, s\ Happens$ *(DisApproveRule*$(r), t) \rightarrow HoldsAt$*(TargetHolds*$(r), t)$
$\&\ [\neg HoldsAt$*(TrustHolds*$(s), t)\ |\ \neg HoldsAt$*(RuleEffectIsPermit*$(r), t)].$

$\forall r, t:\ Happens$ *(RejectRule*$(r), t) \rightarrow HoldsAt$ *(TargetDoesntHold*$(r), t).$

In this model, we have defined the rule sort r, the subject sort s, and some *events* and their corresponding *fluents*. Unlike the policy model which combines several rules, the rule model reasons on rules one-by-one. Each rule is defined by a subject as a set of attributes (a rule instance is given below). Further, like in the policy model, we defined some action precondition axioms. However, a rule's precondition action axioms depends on the *rule-target* (i,e. subject, object and action), the subject's *trust-level* attribute and the *rule-effect*. The latter refers to the state of the delegation-rule, *i.e.*, if it remains valid or not. This will allow the *delegator* to keep control over resources it shares, by deactivating and/or activating the delegation-rule at any time point over the collaborative sessions. This can be done with a basic update on the rule's *effect* attribute at the level of the *CAdS* module.

Thus, a delegation-rule is permitted if: (1) the request attributes match the delegation-rule's target; (2) the requester's trust-level attribute satisfies the threshold set by the organization that hosts the requested resource; (3) the rule-effect is permit (not deny). In the next model ($Model3$) we describe how the *CDS* checks if a rule-target holds (or not).

Model 3 (Rule Target Verification model)

sort r
fluent $[TargetHolds/TargetDoesntHold](r)$
event $[MatchParams/MisMatchParams](r)$

$\forall r, t:\ Initiates(MatchParams(r), TargetHolds(r), t).$
$\forall r, t:\ Initiates(MisMatchParams(r), TargetDoesntHold(r), t).$

$\forall r, t:\ Happens$ *(MatchParams*$(r), t) \rightarrow \neg HoldsAt$ *(TargetHolds*$(r), t).$

$\forall r, t:\ Happens$ *(MisMatchParams*$(r), t) \rightarrow \neg HoldsAt$ *(TargetDoesnt-Hold*$(r), t).$

The target-rule model reasons also on the rules one-by-one, and the events are subjected to some condition action axioms defined at the level of each rule instance, e.g. the rule depicted in the $Model6$. These conditions depend on the matching between the incoming request attributes and the parameters defined in the rule, namely: subject, object, action, *delegator* and *delegator* status (on-line /off-line).

Now, let's get back to the main delegation-rule model $Model2$. One of preconditions action axiom is about the subject's *trust-level* : $HoldsAt(TrustHolds(s), t)$. The latter must be held to approve the delegation-rule. The model for trust-level assessment is depicted in $Model4$:

[7]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)* [14]

Model 4 (Trust Verification model)

$sort \quad s, userTrLevel, orgTHR$
$predicate \quad UserTrlevel(userTrLevel)$
$predicate \quad OrganizationThold(orgTHR)$
$fluent\ [IsTrusted/IsNotTrusted](s)$
$event\ [TrustHolds/TrustDoesNotHold](s)$

$\forall r,t:\ Initiates(TrustHolds(s), IsTrusted(s), t).$
$\forall r,t:\ Initiates(TrustDoesNotHold(s), IsNotTrusted(s), t).$

$\forall r,t,userTrLevel,orgTHR:\ Happens\ (TrustHolds(s),t)$
$\&\ UserTrlevel(userTrLevel)\ \&\ OrganizationThold(orgTHR) \rightarrow$
$orgTHR \geq userTrLevel.$

$\forall r,t,userTrLevel,orgTHR:\ Happens\ (TrustDoesNotHold(s),t)$
$\&\ UserTrlevel(userTrLevel)\ \&\ OrganizationThold(orgTHR) \rightarrow$
$orgTHR < userTrLevel.$

In this model, we have defined the subject sort s, $userTrLevel$ and $orgTHR$. The last two sorts are respectively used by the predicates $UserTrlevel$ and $OrganizationThold$ for checking that the *delegatee* trust-level is higher than (satisfies) the organization's trust-threshold.

Then, $Model5$ is used to check if a given *delegator* is online. We use the predicate $Terminates$ that will set the fluent $IsOnline$ to false after the occurrence of the $Disconnect$ event.

***Model 5** (delegator **Status Verification model**)*

$sort \quad s$
$fluent\ (Online)(s)$
$event\ [Connect/Disconnect](s)$

$\forall s,t:\ Initiates(Connect(s), OnLine(s), t).$
$\forall S,t:\ Terminates(Disconnect(s), OnLine(s), t).$

$\forall r,t:\ Happens\ (Connect(s),t) \rightarrow \neg Online(s).$
$\forall r,t:\ Happens\ (Disconnect(s),t) \rightarrow Online(s).$

C. Application to the motivating example

Because the above logical models are not trivial to write, we have implemented a service [15] at the level of the *CDS* module to automatically generate the delegation-rules patterns based on a set of attributes. Those attributes are defined by *delegators* and organizations at the level of the *CAdS* module. The service also generates the Event-calculus predicates for every *delegatees* request.

The $Model6$ illustrates the first scenario where *Jessy* delegates to *Alice* the updating rights of the association calendar. In this model we have the delegation-rule instance $DelegAlice1$ in which some attributes issued from *Alice*'s access request (depicted at the top of the $Model7$) must match to the $DelegAlice1$'s attributes values. In addition, the rule effect is also set at permit, *i.e.*, fluent $HoldsAt(RuleEffectPermitted(DelegAlice1), 0)$.

Model 6 (Rule pattern example)

$r \quad DelegAlice1$

$\forall t,s,o,a,d:\ Happens(MatchParams(DelegAlice1),t) \rightarrow s = Alice\ \&\ o = Calendar\ \&\ a = PUT\ \&\ d = Jessy\ \&\ \neg HoldsAt(OnLine(Jessy),t).$

$\forall t,s,o,a,d:\ Happens(MisMatchParams(DelegAlice1),t) \rightarrow s \neq Alice\ |o \neq Calendar\ |a \neq PUT\ |d \neq Jessy\ |\ HoldsAt(OnLine(Jessy),t).$

$HoldsAt(RuleEffectPermitted(DelegAlice1),0).$
$\neg HoldsAt(RuleEffectNOTpermitted(DelegAlice1),0).$

The reasoning results showed in $Model7$ are provided by the *DEC-Reasoner*[8].The reasoning process is based on a real time events stream. That means that, at a given time-point, all the environment attributes (*i.e.*, *fluents*) are held at true or false (respectively **"+"** and **"-"** at the left of the fluent), and any event could happen at any time-point. The occurrence of the delegation-models' events will cause a state change on the environment attributes by changing the corresponding *fluents* states. Note that, if a *fluent* state is changed, it will remain unchangeable until the occurrence of another event able to modify it. The events are monitored by the *CES* and are continuously injected in the DEC-reasoner through the *CDS*. Then, when *fluents*' states change, events that are related to precondition-action-axioms that involve the changed *fluents* could happen (or not) over the continuous reasoning time-interval.

Model 7 (Request evaluation example)

#Request inputs:
subject Alice object Calendar action Put delegator Jessy UserTrlevel(7) OrganizationThold(5)

#Reasoning results:
$t = 0$
RuleEffectPermitted(DelegAlice1).
- *OnLine(Jessy).*
Happens(TrustHolds(Alice), 0).
$t = 1$
+ *IsTrusted(Alice).*
Happens(MatchParams(DelegAlice1), 1).
$t = 2$
+ *TargetHolds(DelegAlice1).*
Happens(Epermit(DelegAlice1), 2).
$t = 3$
+RulePermitted(DelegAlice1).

To facilitate the understanding of the reasoning results, we ordered the occurrence of the different events in different time-points in order to separate them for better clarity. We suppose that at time point 0, the *delegator Jessy* is offline. At the same time-point, the event for checking *Alice*'s trust-level happens. Then, the *fluent* that states that *Alice* is a trusted person is held at time-point 1. Next, the same occurs for the rule-target at the time-points 1 and 2. Finally, after all delegation-rule event's preconditions are satisfied, the delegation will be authorized.

This example illustrates only one rule, and due to space limitation, we cannot present another delegation-rule(s) for illustrating the reasoning results on a *Policy*. But, if the latter contains, among other delegation rules, the delegation-rule instance $DelegAlice1$, then, for the same request attributes, the result obtained after the reasoning based on the policy model ($model1$) will approve the request.

V. Trust Assessment

A. Trust assessment background

The other part of our proposal deals with trust. Our trust assessment mechanism is based on the subject's historical data collected over its collaborative sessions. A session is the time-interval used for collecting subjects' *logs*. For every subject, we monitor the evolution of its trust-level regarding its past sessions. Then, based on the evolution, we determine how to assess the subject's behavior during the next session. From a security point of view, we aim to detect the negative behavior

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

performed by a subject. From this perspective, several metrics could be included under the scope of the negative unauthorized behavior, *e.g.*, misuse on access gained data, virus implant, *etc.*. To simplify the assessment we only focused on the illegal access requests. An illegal access request is automatically rejected by the community's access control mechanism. We assume that each subject's access request(s) with the corresponding access control decisions are collected, aggregated and stored within the subject's *log* file.

The main idea of our algorithm is as follows. To compute the trust-level for a given subject – *James* – the *CTS* computes at the end of every session the trust-level evolution rate between *James*' last session and the median of the remaining ones. Consequently, the *CTS* computes the *penalty-factor* to apply on *James*' possible illegal actions collected at the end of his next session. Note that, for each subject, the *penalty-factor* is evolutive. This means that, it may change at every session (*i.e.*, it may increase, decrease or remain stable) depending on the evolution of the subject's past trust-level.

B. Trust assessment formalization

In our context, we consider that the subject's trust-level decreases and/or increases at a rate proportional to the number of its illegal actions, *i.e.*, number of denied requests. Further, in our context, the trust assessment process allows us to enhance the basic access-control mechanism in the collaborative environment. Therefore, a rapid interception of a suspicious subject's behavior is required to be taken into account for the access-control process. Therefore, we formalize the trust-assessment scheme with an *exponential* function that take the subject's parameters: number of denied requests $NbDeny$ and the last penalty-factor ρ (as described in **algorithm 1**).

Algorithm 1 Trust assessment

1: **function** $\mathbf{TrustComputing}(\rho,\ nbDeny)$
2: $\quad tr = \exp(-\rho * nbDeny);$
3: $\quad$ **return** tr;
4: **end function**

As we said above the penalty-factor ρ value may change after the end of every session. The increase (or decrease) of ρ will allow a subject to pay attention to its behavior. This behavior has a direct impact on the evolution of ρ as when a subject doesn't try to access unauthorized resource(s), the applied ρ will remain low.

To compute subject's ρ that will be applied for its next session, we first need to compute the subject's initial trust-level tr_0 based on its historical past experiences stored by the *CIS*. Then, we need to measure the evolution rate λ between the initial trust-level and the one tr computed at the end of the subject's last session.

Let's assume that n is the number of a subject's collaborative sessions. To compute tr_0, the *CTS* computes the average of the $(n-1)$ subject's historical trust values. However, the *CTS* gives more importance (weighting it by 2) to the last experience $(n-1)$ regarding the remaining ones $(n-2)$.

Then, to compute the trust evolution-rate λ, the *CTS* measures the change rate between tr_0 and tr using the exponential decay/growth function. Thus, if tr is higher than tr_0, then, λ will be a positive value. Otherwise, λ will be a negative value. Consequently, if the evolution is positive (*i.e.*, $\lambda > 0$), it will narrow down the penalty-factor ρ. Conversely, ρ will arise. However, an untrusted subject must not be able to increase its penalty-factor as fast as a trusted subject. On the other hand, a trusted subject must be penalized with more severity than an untrusted subject. This means that the subject's trust category [33] interpreted by its last penalty-factor has a direct impact on the ρ evolution (more details below). Therefore, we weight the obtained λ with the reverse of the subject's last penalty-factor, (*i.e.*, $1-\rho$). Further, we need also to soften the sudden high change of λ, so, we divide the obtained result by the *community-severity-factor* ς. The latter is a subjective parameter defined by the community administrator. The higher this parameter is, the harder for subjects to decrease (or increase) their respective ρ is.

Algorithm 2 trust evolution factor

1: **function** $\mathbf{TrEvolFact}(tr_0,\ tr,\ \varsigma,\ \rho)$
2: $\quad \lambda = [(\ln(tr/tr_0)/2) * (1-\rho)]/\varsigma;$
3: $\quad$ **return** λ;
4: **end function**

As we said above, the computation of new penalty-factor ρ is based on the result of the last evolution-rate λ. Therefore, we have two kinds of penalty-factors, the *discrete* penalty-factor ρ and the *continuous* penalty-factor ϱ. The latter ϱ is used to analyze the continuous changes of a subject's behavior, while the former ρ is applied for penalizing the subject's possible illegal actions. Then, we compute the penalty-factor ρ as follows. First, we subtract the current trust evolution rate λ from the last continuous penalty-factor ϱ. Then, we discretize the obtained ϱ value based on the discrete community interval ($DiscP[\,]$). The latter is one of the parameters the community creator defines during the community creation process to specify the categories of subjects' trust, e.g. *full-trusted, trusted* or *untrusted person*. Each label corresponds to a real value. Note that, the closer the discrete values are, the less the trust-values drop after undesired subjects' behaviors.

Algorithm 3 trust penalty-factor

1: **function** $\mathbf{penaltyFactor}(\varrho,\ \rho,\ \lambda)$
2: $\quad \varrho = \varrho - \lambda;$
3: $\quad \rho = \mathbf{PenaltyDiscritization}(\varrho, DiscP[\,]);$
4: $\quad$ **return** ρ;
5: **end function**

According to our λ computing function, the penalty-factor's discrete values must always be $\in]0,1[$. Because if ρ is equal to zero, then, no penalty is applied in the community. However, if is equal to 1, the evolution rate will converge to be equal to zero in the upper bound. That's why it is recommended to use a discrete interval $DiscP\,[\,] \subseteq]0,1[$, except if the trust assessment would be intentionally deactivated.

For the $PenaltyDiscritization$ procedure, we use the *binary search* algorithm to look for the nearest discrete value in the community discrete interval regarding the computed penalty-factor ρ. The discretization procedure is used to prevent subjects to change immediately their penalty-factor after a collaborative session. In addition, when ϱ value exceeds the lower and/or upper bound of the discrete set, ρ value is

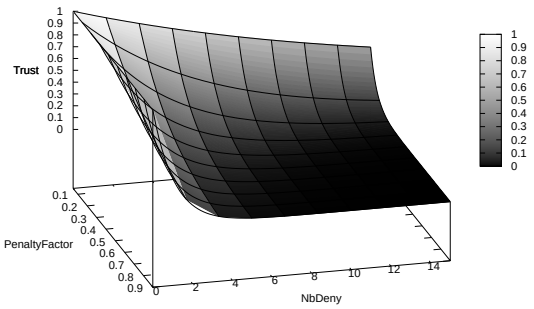

Fig. 3: Trust assessment function.

consequently reduced to the $DiscP[]$'s lower and/or upper bounds respectively. Therefore, the same penalty value will be applied to all subjects of a given community.

C. Illustration

For illustrative purposes, we use the following discrete interval $DiscP[] = [0.05, 0.1, 0.5, 0.9]$, which can be interpreted as the following discrete labels: *Very Trustworthy, Trustworthy, Untrustworthy, Very Untrustworthy* based on [33] trust model. Furthermore, we suppose that the maximum accepted unauthorized access attempts before a requester is suspended to be $maxCommNbDeny = 15$. The results of such settings with the use of our approach for trust computing are depicted in the figure, Fig. 3. As we can notice the figure, the more penalty-factor ρ will increase, the more the trust level will decrease towards the increasing of the number of the unauthorized requests.

VI. Experimentation

To evaluate our trust assessment approach, we have made some tests based on two subject experiences, namely, *Alice* and *Oscar*. This service was implemented in the *OpenPaaS* project[9]. However, we currently lack real users' relevant benchmarks. Thus, the simulation done shows the various impacts on the trust-level.

A. Presentation

The session duration for collecting logs is subjective, and depend on the community administrator (*i.e.*, every hour, day, month, trimester or more). In addition, the number of the denied requests collected after each session is supposed to be significant, relevant and not corrupted. In this work, we do not tackle those challenges, however, they could be handled with machine learning approaches based on subjects' experiences, that in a real case scenario, will depend on several parameters, like session spent time, interaction frequency, *etc*. Future work will be made in this concern with the use of a real experimental subjects Logs. Moreover, In this work we don't address the problem of initializing new subject's parameters, i.e., the one who has joined a collaborative session. We instead attribute average values, *e.g.*, $0.4, 0.5, 0.6$.

[9]https://research.linagora.com/display/openpaas/Open+PaaS+Overview

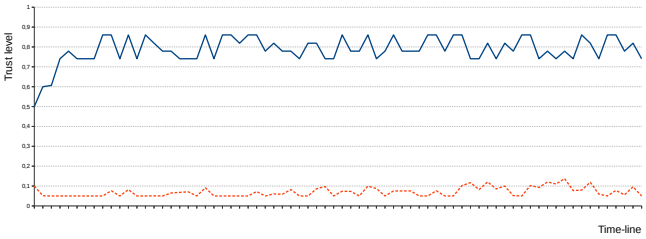

Fig. 4: Trust evolution for consistent behavior.

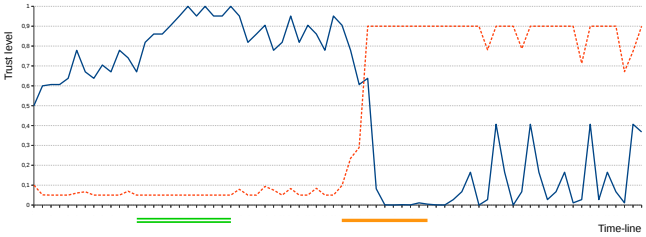

Fig. 5: Trust evolution for inconsistent behavior.

We consider that our two subjects *Alice* and *Oscar* start with the same initial values, namely, $tr_0 = 0.5$, $tr = 0.6$, $\rho = 0.1$, $\varrho = 0.1$. To simplify the illustration of the experimental results, we have choose a *community-severity-factor* ς equal to 1, which means that the ρ evolution will be fast. The dashed graph shows the evolution of the *penalty-factor* ρ, whereas, the plain one shows the *trust-level* evolution. We consider *Alice* as a trusted person with a consistent behavior. Therefore, *Alice*'s behavior is a vector of random values between 3 and 6 $nbDeny$. On the other hand, *Oscar*'s behavior is inconsistent. We have defined four different behavior vectors for *Oscar*. In the first one, *Oscar* has a normal behavior for a given session period. Then, for the next period, *Oscar* will have an exemplary behavior, which will normally decrease his penalty-factor. To check this, for the third period *Oscar* will switch to the normal behavior. Further, *Oscar*'s undertakes a suspicious behavior by a high frequency of illegal access requests. Therefore, after such behavior *Oscar* must be penalized. Then, for the fourth period, *Oscar* will switch again to the normal behavior *Normal2* which is better than the first one (in term of the number of rejected requests).

Normal	5	10	9	5	8	9	7	8	5	6	8	
Normal2	4	3	2	5	4	1	4	2	3	5	1	2
Good	4	3	5	2	2	1	1	0	2	1	1	
Bad	10	4	3	2	14	4	1	2	11	3	1	2

TABLE I: *Oscars*'s behaviors

B. Discussion

Fig 5 shows the *Oscar*'s trust evolution over sessions. The double-line fragment shows the good period, whereas, the single-line one shows the bad period. Thus, as we can see Fig5 graphic, after the good period, *Oscar*'s trust level increases because, for the same behavior (*Normal1*), he obtains better trust scores than the initial ones.

On the other hand, Fig 4 shows *Alice*'s trust evolution. As *Alice* has a stable more-or-less good behavior, her trust scores remain stable over the collaborative sessions.

VII. Conclusion

In this paper, we proposed a formal delegation framework designed for enterprise social networks to allow subjects to delegate access to their resources when they can not be available. Our Event-Based delegation model aims to reason about the delegation events that happens within the collaborative environment to manage the delegation's access-control policies.

This delegation model is used under a central federated architecture to deal with the heterogeneity issues within collaborative platforms. However, collaborative organizations are not obliged to fully delegate the access control management to a central third party. The traditional delegation access control approaches are either user-centric or organization-centric and we have advocated to bridge the gap between them. Because, in our architecture, organizations are able to keep control over their resources (*i.e.*, autonomy) by means of fine grained constraints concerning the required external principals' trust-level. Those constraints are combined with the users' access control delegation rules, and are computed based on a real-time monitoring of principals' behaviors. For this, we have proposed an efficient trust assessment mechanism and showed the trust-level variation results based on simulated users' behaviors.

For future work, we aim to improve our delegation model by considering, delegation of co-owned resources and revocation, timed and multi-step delegation [16], and try to get a real relevant benchmarks about users' behaviors, in order to validate our trust assessment approach with better experimental evaluation.

References

[1] R. S. Sandhu and P. Samarati, "Access control: principle and practice," *Communications Magazine, IEEE*, vol. 32, no. 9, pp. 40–48, 1994.

[2] T. Issa and P. Kommers, "Social networking for web-based communities," *International journal of web based communities*, vol. 9, no. 1, pp. 5–24, 2013.

[3] S. Mithas, T. Costello, and A. Tafti, "Social networking in the enterprise," *IT Professional*, vol. 13, no. 4, pp. 16–17, 2011.

[4] D. Koren, "Social networking for the police enterprise," 2013.

[5] S. S. Shim, G. Bhalla, and V. Pendyala, "Federated identity management," *Computer*, vol. 38, no. 12, pp. 120–122, 2005.

[6] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The ponder policy specification language," in *Policies for Distributed Systems and Networks*. Springer, 2001, pp. 18–38.

[7] V. C. Hu, D. R. Kuhn, and D. F. Ferraiolo, "Attribute-based access control," *Computer*, no. 2, pp. 85–88, 2015.

[8] L. Zhang, G.-J. Ahn, and B.-T. Chu, "A role-based delegation framework for healthcare information systems," in *Proceedings of the seventh ACM symposium on Access control models and technologies*. ACM, 2002, pp. 125–134.

[9] K. Gaaloul, H. A. Proper, and F. Charoy, "Delegation protocols in human-centric workflows," in *Commerce and Enterprise Computing (CEC), 2011 IEEE 13th Conference on*. IEEE, 2011, pp. 219–224.

[10] J. A. Russo and R. Yates, "Time limited collaborative community role delegation policy," Aug. 19 2008, uS Patent 7,415,498.

[11] S. S. Yau, Y. Yao, and A. B. Buduru, "An adaptable distributed trust management framework for large-scale secure service-based systems," *Computing*, vol. 96, no. 10, pp. 925–949, 2014.

[12] J. Carter, E. Bitting, and A. A. Ghorbani, "Reputation formalization for an information–sharing multi–agent system," *Computational Intelligence*, vol. 18, no. 4, pp. 515–534, 2002.

[13] L. Kagal, T. Finin, and Y. Peng, "A delegation based model for distributed trust," in *Workshop on Autonomy, Delegation, and Control: Interacting with Autonomous Agents, International Joint Conferences on Artificial Intelligence*, 2001.

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

[15] E. Zahoor, O. Perrin, and A. Bouchami, "CATT: A cloud based authorization framework with trust and temporal aspects," in *10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing, CollaborateCom 2014, Miami, Florida, USA, October 22-25, 2014*, 2014, pp. 285–294. [Online]. Available: http://dx.doi.org/10.4108/icst.collaboratecom.2014.257312

[16] E. Barka, R. Sandhu *et al.*, "A role-based delegation model and some extensions," in *23rd National Information Systems Security Conference*. Citeseer, 2000, pp. 396–404.

[17] E. Barka and R. Sandhu, "Role-based delegation model/hierarchical roles (rbdm1)," in *Computer Security Applications Conference, 2004. 20th Annual*. IEEE, 2004, pp. 396–404.

[18] E. Barka and R. S. Sandhu, "Framework for role-based delegation models," in *16th Annual Computer Security Applications Conference (ACSAC 2000), 11-15 December 2000, New Orleans, Louisiana, USA*, 2000, p. 168. [Online]. Available: http://dx.doi.org/10.1109/ACSAC.2000.898870

[19] L. Zhang, G.-J. Ahn, and B.-T. Chu, "A rule-based framework for role-based delegation and revocation," *ACM Transactions on Information and System Security (TISSEC)*, vol. 6, no. 3, pp. 404–441, 2003.

[20] Z. Liu, W. Sun, and M. Alam, "A flexible role-based delegation model with dynamic delegation role structure."

[21] P. Tsankov, S. Marinovic, M. T. Dashti, and D. Basin, *Decentralized composite access control*. Springer, 2014.

[22] L. Seitz, E. Rissanen, T. Sandholm, B. S. Firozabadi, and O. Mulmo, "Policy administration control and delegation using xacml and delegent," in *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*. IEEE Computer Society, 2005, pp. 49–54.

[23] B. S. Firozabadi, M. Sergot, and O. Bandmann, "Using authority certificates to create management structures," in *Security Protocols*. Springer, 2002, pp. 134–145.

[24] K. Gaaloul, E. Zahoor, F. Charoy, and C. Godart, "Dynamic authorisation policies for event-based task delegation," in *Advanced Information Systems Engineering*. Springer, 2010, pp. 135–149.

[25] A. Jsang and R. Ismail, "The beta reputation system," in *Proceedings of the 15th bled electronic commerce conference*, 2002, pp. 41–55.

[26] L. Mui, M. Mohtashemi, and A. Halberstadt, "A computational model of trust and reputation," in *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*. IEEE, 2002, pp. 2431–2439.

[27] S. Buchegger and J.-Y. Le Boudec, "A robust reputation system for peer-to-peer and mobile ad-hoc networks," in *P2PEcon 2004*, no. LCA-CONF-2004-009, 2004.

[28] W. L. Teacy, J. Patel, N. R. Jennings, and M. Luck, "Travos: Trust and reputation in the context of inaccurate information sources," *Autonomous Agents and Multi-Agent Systems*, vol. 12, no. 2, pp. 183–198, 2006.

[29] V. Cahill, "Using trust for secure collaboration in uncertain environments," 2003.

[30] V. Sassone, K. Krukow, and M. Nielsen, "Towards a formal framework for computational trust," in *Formal Methods for Components and Objects*. Springer, 2007, pp. 175–184.

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

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

[33] A. Abdul-Rahman and S. Hailes, "Supporting trust in virtual communities," in *System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on*. IEEE, 2000, pp. 9–pp.