MuAC: Access Control Language for Mutual Benefits*

Lorenzo Ceragioli[1], Pierpaolo Degano[1], and Letterio Galletta[2]

[1] Università di Pisa, Pisa, Italy
lorenzo.ceragioli@phd.unipi.it, degano@di.unipi.it
[2] IMT School for Advanced Studies, Lucca, Italy
letterio.galletta@imtlucca.it

Abstract

In a collaborative distributed environment, users own a set of private resources that they possibly share with each other to achieve mutual advantages. The access to resources is regulated by a policy defined by each user in isolation, and independently of the others. However, typical access control languages allow defining policies that only check the roles or the attributes of the requesters and resources. But they do not impose a fair exchange of access grants by taking into account what requesters offer to others. Here, we present MuAC, a logic-based access control language designed for expressing mutuality. In particular, *MuAC* allows specifying conditions on what requesters must offer in exchange for using a particular resource.

1 Introduction

Access control mechanisms selectively restrict accesses to resources, distinguishing between legit and illicit requests. Here we will focus on the case where the permitted accesses are described via a policy, expressed in a suitable language (policy-based access control). Different such languages exist, from low level ones, e.g. those used in firewalls like IPTABLES [13], to high level ones like XACML [8].

In a distributed setting each user has a set of his own resources, that are possibly shared with others. The access policy protecting these resources is naturally defined by each user in isolation, and independently of the other users. In distributed collaborative contexts, policies are designed to achieve a fair exchange of access grants, so enhancing mutual advantages. For example, different hospitals may share anonymized medical data to improve the quality of statistics. A further example are online social networks where interactions are regulated: Alice decides on her own how her pictures can be accessed, e.g. by only showing them to who shares his pictures with her. As a matter of fact, mutuality is the basis of social interactions, and it affects how resources are managed and shared.

Traditional access control languages do not express such conditions that foster mutual benefits, but only check the roles or the attributes of the requesters, of the resources and also of the environment. Typically, the exchange of access rights is negotiated by humans and implemented by hand in the access control policy of each contractor. An automatic tool would instead help, which takes access control decisions on what requesters offer to others. To the best of our knowledge, only few papers have investigated the possibility of expressing a limited form of mutuality in policies, e.g. [12].

*The first two authors have been partially supported by project PRA_2018_66 *DECLware: Declarative methodologies for designing and deploying applications* of the Università di Pisa; the third author by MIUR project PRIN 2017FTXR7S *IT MATTERS* (Methods and Tools for Trustworthy Smart Systems).

Here, we start to address the mutuality issue and propose *MuAC*, a logic-based access control language. This language not only allows specifying conditions on the resource and users, but also constraints on what requesters must offer in exchange. We claim that *MuAC* naturally expresses situations in which mutuality has involved forms.

Even though policies are exclusive of each user, they impact on each other. For example, consider the case of picture sharing mentioned above: Alice needs to be aware of Bob's policy for granting him access to her pictures. In general, deciding an access request may require a complete knowledge of the policies of every user. Here, we assume a centralized evaluation model where each user separately defines his local policies and sends them to the policy enforcement point (PEP), responsible for the evaluation of the requests and the interactions among users.

Mutuality may however induce circularity while evaluating access requests. Consider again picture sharing and assume that the PEP receives from Bob a request to watch Alice's pictures. If Bob unconditionally allows Alice to watch his, then there is no circularity and Bob's request is accepted. A circularity instead occurs if Bob allows Alice to watch his picture *only if* she shares her pictures with him. Nevertheless, the PEP should grant Bob access to Alice's pictures, and viceversa.

Unfortunately, classical logic is not fully adequate to express this situations, which are typical of common contracts. To overcome this limitation, an effective predicative contractual logic (PCL) was introduced in [2] that extends intuitionistic predicate calculus with the operator $\twoheadrightarrow$, called *contractual implication*. The formula $A \twoheadrightarrow B$ intuitively stands for a promise that "B will be satisfied if A is as well".

We therefore exploit PCL to resolve circularity arising in mutual access policies. More in detail, the PEP translates the policies of the users into a conjunction of PCL formulas, and relies on teh deductive machinery of [1] to allow or not a given request.

Plan of the paper Section 2 gently introduces *MuAC* and its features through a running example. Section 3 briefly reviews the logic PCL. Section 4 presents the syntax and the semantics of *MuAC*; discusses its expressive power through a bunch of examples; and describes the algorithm implementing a centralized PEP for request evaluation. Section 5 compares our proposal with the literature, and draws some conclusions and future work.

2 Example: Sharing Resources among Research Groups

Consider a computer science department with several research group on different areas. Each group has a finite amount of resources, some of which may be shared to enhance cooperation and to optimize their usage. Sharing however is not always for free, and each research group wants something in return for allowing a fair access to its resources, while avoiding free raiders and guaranteeing mutuality of the benefits. In other words, each researcher should be enabled to decide which resources to share and with whom, and which guarantees to require, so giving rise to an access policy. We here offer a formal language for specifying individual policies, independently of each other. The task of the automatic access control system we propose is to combine these policies together and enforce them on all the involved entities.

We now informally describe the department; then we intuitively present possible individual policies and their formalization in a simplified form; and finally we discuss how the individual policies are combined to control accesses.

There are three research group inside the department: machine learning, consisting of Mark, Michelle and Morty; networks, including Nick, Nancy and Neil; and security, with Selene and Sam. Resources are of different kinds, e.g. `computational power`, `data`, `software`, `picture`.

Table 1 describes what researchers require (line *wants*), what they offer (line *shares*) and under which conditions (line *with*). For example, Morty (i) looks for `picture` and offers in exchange `computational power`, provided that who requires it shares something with him, in this case `pictures`; moreover, he (ii) offers for free his `software`. Nick offers `pictures` and `post`, under the condition that the requester is in his group of shares something with its member, and wants `computational power`. Selene instead requires both `network software` and `source code` and offers `network logs` to the members of her group who share something with her, or also to a member who shares something with someone sharing with her (a sort of transitive sharing).

More formally, the case (i) of Morty's policy is rendered by the following rule

$$\mathit{computational\text{-}power}(\texttt{Resource}),\ \mathit{Allows}(\texttt{Me},\texttt{r},\texttt{Subject}) \tag{1}$$

where the first predicate is true when the resource requested is of kind `computational power`. The predicate *Allow* is typical of our policy language and is the key one. It has three arguments: the first is the individual who decided the policy, here `Me` is Morty; the second argument is the resource requested by Morty; and the last one says that r is owned by `Subject` (note that the check that the type of r is *picture* is completely demanded to the access control manager). Intuitively, *Allow* evaluates true when `Subject` permits `Me` to access `r`. But this cannot be established without inspecting the individual policy of `Subject`.

The case (ii) of Morty's policy, i.e. he shares his software with everyone, is easier:

$$\mathit{software}(\texttt{Resource})$$

As expected, Morty and Nick can share some of their resources. Indeed, the relevant portion of Nick's policy is rendered the following formula:

$$\mathit{pictures}(\texttt{Resource}),\ \mathit{Allows}(\texttt{s},\texttt{r},\texttt{Subject}),\ \mathit{Networks}(\texttt{s}) \tag{2}$$

Note that the third predicate constrains Morty to share something with a member of the Networks group, here Nick himself. The sharing between Morty and Nick can only take place if both (1) and (2) hold. This happens when in the rules

(1) `Me` stands for Morty; `Subject` is bound to Nick; and `r` to a picture P offered by Nick.

(2) `s` is bound to Nick; `Subject` stands for Morty; and `r` is bound to `computational power` CP offered by Morty.

As mentioned above, the predicate *Allows* plays a key role and deserves a non standard treatment. Indeed, in order to prove that *Allows*(Morty, P, Nick) holds, we need to show that *Allows*(Nick, CP, Morty) holds, and viceversa so leading to a circularity. To break it safely, we exploit the contractual implication of Propositional Contract Logic [2], through the construction detailed in the next section.

3 Background: Propositional Contract Logic

Bartoletti e Zunino introduced in [2] the intuitionistic propositional logic of contracts, whose distinguishing operator is *contractual implication* $\twoheadrightarrow$. Intuitively, the formula $p \twoheadrightarrow q$ means "q is granted to be true if p holds as well." An alternative interpretation is "a promise that q will be satisfied if also p is." Consider the social network example of Section 1, if Alice ensures that Bob can see her pictures if Bob does the same is rendered by *Alice-sees-Bob* $\twoheadrightarrow$ *Bob-sees-Alice*.

Table 1: What researchers require and offer

MACHINE LEARNING GROUP			
	Mark	**Michelle**	**Morty**
wants:	source code	network logs	pictures
shares:	computational power	computational power	computational power
with:	who shares with him	who shares with her	who shares with him
shares:	software	software	software
with:	everybody	everybody	everybody

NETWORKS GROUP			
	Nick	**Nancy**	**Neil**
wants:	computational power	computational power	networks logs
shares:	pictures	network software	
with:	group, who shares with group	group, who shares computational power with group	
shares:	posts		
with:	group, who shares with group		

SECURITY GROUP		
	Selena	**Sam**
wants:	network software, computational power	source code
shares:	network logs	source code
with:	group member sharing with her or group member sharing with someone who shares with her	who shares with him or who shares with someone who shares with him

The main aspect of contract logic is its ability of dealing with the circularity arising when p is verified if q is as well and vice versa, formally $(p \twoheadrightarrow q) \wedge (q \twoheadrightarrow p)$. In contrast with common implication, here the result is that both the formulas are verified. This handshake property is formally stated as

$$\vdash (p \twoheadrightarrow q) \wedge (q \twoheadrightarrow p) \to p \wedge q$$

Of course, circularity may involve many steps, rather than the two considered above.

Contractual implication satisfies transitivity

$$\vdash (p \twoheadrightarrow q) \wedge (q \twoheadrightarrow r) \to (p \twoheadrightarrow r)$$

A further natural property is that promises in contracts can be arbitrarily weakened, while the precondition can be arbitrarily strengthened. Intuitively: if I am willing to grant you something in change of something else, then I am also willing to give you less for getting more:

$$(p \twoheadrightarrow q) \wedge (q \to q') \to (p \twoheadrightarrow q') \qquad (p' \to p) \wedge (p \twoheadrightarrow q) \to (p' \twoheadrightarrow q)$$

Finally, a contract should be satisfied also when the premise is unconditionally satisfied, and if a promise q is already true, then it is also true any contract which promises q

$$\vdash p \wedge (p \twoheadrightarrow q) \to q \qquad \vdash q \to (p \twoheadrightarrow q)$$

The propositional intuitionistic logic has been proved decidable. The theorem prover of [1] implements the PCL entailment, the performance of which is acceptable for common examples, although in general deduction is PSPACE complete.

4 The *MuAC* Access Control Language

Our model considers a system hosting a finite set $\mathcal{U}$ of users (ranged over by u, u', u_i) that protect and use a finite set $\mathcal{R}$ of resources (ranged over by r, r', r_i) of various kinds. Each user chooses how to regulate accesses to his resources by specifying a set of rules, describing the conditions under which access is granted to another user, called `Subject`. Although the specification of access rights is fully demanded to each users, their enforcement results from the combination of these individual policies and a centralized entity grants or denies access, upon receiving a request.

Policy specification A rule $\phi \in \Phi$ contains atomic propositions on users and resources, and possibly the special predicate *Allows*. We assume a set of user variables $\mathtt{u}, \mathtt{u}' \ldots$; a set of resource variables $\mathtt{r}, \mathtt{r}' \ldots$; three special variables `Me`, `Subject`, `Resource` that take value when used (`Me` indicates the owner of the rule, `Subject` the requester and `Resource` has the obvious meaning); and a set of atomic predicates $p, q, p', q' \ldots$ Let U range over `Me, Subject` and user variables; and let R range over `Resource` and resource variables. Rules are defined as follows

$$\phi ::= p(U) \mid p(R) \mid \mathit{Allows}(U, R, U) \mid \phi, \phi$$

The intended meaning of a rule ϕ is that `Me` grants `Subject` access to `Resource` if all the predicates occurring in ϕ are proved true, possibly involving circularity.

Atomic predicates express properties of users and resources that are meaningful in the overall system. E.g., *software*(`Resource`) in Morty's policy holds true if the requested resource is a piece of software. The predicate $\mathit{Allows}(U, R, U')$ states the condition that the user U is allowed by user U' to access resource R. An example is the formula *Allows*(`Me`, `r`, `Subject`), occurring in the rule (1) of Morty's policy. The comma is the standard logical conjunction. We assume that all the variables are existentially quantified, except for the special ones. See for instance the discussion at the end of Section 2, where the rule (2) of Nick's policy *pictures*(`Resource`), *Allows*(`s`, `r`, `Subject`), *Networks*(`s`) was shown to hold for specific values of the variables `s` and `r`.

Finally, the centralized granting authority collects all the rules in a *configuration*, i.e. a mapping from users to sets of rules $\sigma : \mathcal{U} \to 2^{\Phi}$.

Expressivity Despite the simplicity of the examples considered, *MuAC* allows a user to define involved policies, some of which we discuss below.

In the simplest kind of rule, its owner, i.e. `Me`, offers a resource to the requester, i.e. `Subject` provided that in exchange it shares some (other) resource. This is the case (of a portion) of Sam's policy (3) as intuitively defined in Table 1 speaking both of code, and of Morty's (4) where the exchanged resources are of different nature:

$$\textit{source-code}(\mathtt{Resource}),\ \mathit{Allows}(\mathtt{Me}, \mathtt{r}, \mathtt{Subject}),\ \textit{source-code}(\mathtt{r}) \tag{3}$$

$$\textit{source-code}(\mathtt{Resource}),\ \mathit{Allows}(\mathtt{Me}, \mathtt{r}, \mathtt{Subject}),\ \textit{picture}(\mathtt{r}) \tag{4}$$

Of course, more than one resource can be requested. For example `Me` offers network logs to those who are willing to share both network software and computational power:

$$\textit{network-log}(\mathtt{Resource}),\ \mathit{Allows}(\mathtt{Me}, \mathtt{r}, \mathtt{Subject}),\ \mathit{Allows}(\mathtt{Me}, \mathtt{r}', \mathtt{Subject}),$$
$$\textit{network-software}(\mathtt{r}),\ \textit{computational-power}(\mathtt{r'})$$

Note that to offer k sorts of resources, one has to write k distinct rules.

A second kind of rules considers groups of users rather than a single one: `Me` offers a resource to a member of a group, provided that (another) member gives something in exchange. The following example says that `Me` offers computational power to the security group members if any of them shares some source code with him

$$\begin{aligned}&\textit{computational-power}(\texttt{Resource}),\ \textit{Allows}(\texttt{Me},\texttt{r},\texttt{u}),\ \textit{source-code}(\texttt{r}),\\&\textit{security}(\texttt{Subject}),\ \textit{security}(\texttt{u})\end{aligned}$$

Also the converse is possible, when the shared resource is for a member of the group of `Me` (see also the rule 2):

$$\begin{aligned}&\textit{computational-power}(\texttt{Resource}),\ \textit{Allows}(\texttt{u},\texttt{r},\texttt{Subject}),\\&\textit{networks}(\texttt{u}),\ \textit{networks}(\texttt{Me}),\ \textit{computational-power}(\texttt{r})\end{aligned}$$

So far a single *Allow* occurred in the rules. When there are more, we obtain a sort of transitivity of offers and requests. A simple example follows where all the resources are code, `r` belongs to `Subject` and $\texttt{r}'$ to the user `u`. Intuitively, the rule says that `Me` grants `Subject` access to `Resource` if `Subject` offers `r` to `u`, who in turn offers $\texttt{r}'$ to `Me`:

$$\textit{source-code}(\texttt{Resource}),\ \textit{Allows}(\texttt{u},\texttt{r},\texttt{Subject}),\ \textit{Allows}(\texttt{Me},\texttt{r}',\texttt{u}),\textit{source-code}(\texttt{r}),\ \textit{source-code}(\texttt{r}')$$

Of course this form of transitivity may involve arbitrary chains of *Allow*, as in the rule below, where the second *Allow* is only applied to variables:

$$\begin{aligned}&\textit{source-code}(\texttt{Resource}),\ \textit{Allows}(\texttt{u},\texttt{r},\texttt{Subject}),\ \textit{Allows}(\texttt{u}',\texttt{r}',\texttt{u}),\textit{Allows}(\texttt{Me},\texttt{r}'',\texttt{u}'),\\&\textit{source-code}(\texttt{r}),\ \textit{source-code}(\texttt{r}'),\ \textit{source-code}(\texttt{r}'')\end{aligned}\tag{5}$$

A special form of (multi-user) cooperation is also expressible, illustrated by the following rule. `Subject` is granted access to the picture `Resource` by `Me`, if both `Me` and `Subject` offer to `u` some source code ($\texttt{r}'$ and `r`, resp.):

$$\textit{picture}(\texttt{Resource}),\ \textit{Allows}(\texttt{u},\texttt{r},\texttt{Subject}),\ \textit{Allows}(\texttt{u},\texttt{r}',\texttt{Me}),\textit{source-code}(\texttt{r}),\ \textit{source-code}(\texttt{r}')$$

Policy meaning The semantics of our language is defined by giving meaning to configurations. This is done by first translating a configuration to a finite set Γ of contract logic propositions [2]. Then, the requested access is granted if Γ entails it, according to the proof system of PCL.

We assume that PEP knows which are the resources each user handles, $\mathcal{R}_{\uparrow u}$, and those he wants, $\mathcal{R}_{\downarrow u}$. Also, we assume that this authority can evaluate the atomic predicates appearing in the rules.

We associate with a given configuration σ the PCL theory Γ with propositions defined through the inference rule below, for all $s,u\in\mathcal{U}, r\in\mathcal{R}, \phi\in\sigma(u), \theta\colon (U\to\mathcal{U})\cup(R\to\mathcal{R})$

$$\begin{gathered}\phi_{Allow}[\theta]\twoheadrightarrow \textit{Allows}(s,r,u)\in\Gamma\\ \text{if and only if}\\ \phi_{constraints}[\theta]\wedge\theta(\texttt{Me})=u\wedge\theta(\texttt{Subject})=s\wedge\theta(\texttt{Resource})=r\in\mathcal{R}_{\uparrow u}\cap\mathcal{R}_{\downarrow s}\wedge\\ \forall\text{ predicate }\textit{Allows}(s',r',u')\text{ occurring in }\phi_{Allow}[\theta],\text{ it is } r'\in\mathcal{R}_{\uparrow u'}\cap\mathcal{R}_{\downarrow s'}\end{gathered}$$

where ϕ_{Allow} and $\phi_{constraints}$ are the conjunctions of all the instances of the predicate *Allows* and of the instances of atomic predicates in ϕ, resp., and $[\theta]$ extends homomorphically θ.

A few comments are in order. The ground substitution θ binds in all possible ways the resource and user variables, so to evaluate the atomic predicates in ϕ, except then the special one *Allow*. The predicate $\phi_{constraints}[\theta]$ holds true whenever θ maps the variables to users and resources that satisfy the relevant predicates in ϕ. Furthermore θ is required to bind in the contractual implication the special variables `Me` to the actual owner u of the access rule ϕ, and `Subject, Resource` to the user s and the resource r, belonging to u and requested by s. In addition, all the resources r' occurring in a predicate $Allows(s', r', u')$ should belong to u' and be wanted by s', as well. If all the above holds, $\phi_{Allow}[\theta] \twoheadrightarrow Allows(s, r, u)$ is a formula of Γ.

Policy and request evaluation The Algorithm 1 implements the declarative construction of a PCL theory of the previous paragraph. It presents the function TRANSLATE-INTO-PCL that given a configuration σ processes each rule ϕ therein by computing the legal substitutions θ such that the instantiated atomic predicates evaluate to true. Finally, it collects the obtained formulae into the theory Γ.

The PEP executes Algorithm 2 that receives the policies from the users and builds a configuration σ. Then it translates σ into a PCL theory Γ, using Algorithm 1, and starts serving requests from the users. When the user s tries to access a resource r, his query `asks(s,r)` is transformed into the formula $Allows(s, r, u)$, where u is the owner of r. Finally, the enforcement and decision point checks if the theory Γ entails this formula, i.e. $\Gamma \vdash Allows(s, r, u)$, and replies accordingly. The proof of the entailment is demanded to the deduction system of PCL [2].

It is worth noting that the order in which the access rules are specified is immaterial, because the semantics transforms them in the logical theory Γ. Also, there are neither rules explicitly denying accesses, nor constraining requests to be denied. Consequently, there are no conflicts, in logical terms contradictions, and a permission is granted only when Γ entails the query.

5 Conclusion

We have defined *MuAC*, a control access language for a distributed collaborative environment. Its main feature concerns expressivity, as it permits to define and handle access requests that demand mutual agreement between several users. To resolve the circularities that may arise with this form of mutuality we resorted to the propositional contract logic PCL [2]. Actually, we have defined a translation from policies and requests to logical formula that are then evaluated by the (complete) theorem prover of PCL.

For the time being, *MuAC* has the grant *accept* only. We plan to extend the language with deny rules, which also require to address the resolution of potential conflicts. A further extension is allowing rules in which a user specifies resources that must not be shared. For example, Alice permits Bob to access her pictures, provided that Bob does not share anything with Charlie. This kind of negative requirement is the first step towards the definition of policies regulating conflicts of interest. Finally, we plan to study the relationship between *MuAC* and languages for trust negotiation and the security properties that our language can offer when some users misbehave.

Related work Access control is a wide sudied field, surveyed by many papers, e.g. [9, 11, 14]. Here we only consider discretionary access control [10] because it is a natural choice in distributed cooperative setting, where users individually decide the policies for the resources

Algorithm 1

Input: Configuration $\sigma : \mathcal{U} \to 2^{\Phi}$
Output: PCL theory Γ

function TRANSLATE-INTO-PCL(σ)
  $\Gamma \leftarrow \emptyset$
  for all users $u \in \mathcal{U}$ and rules $\phi \in \sigma(u)$ **do**
    for all user variable $\mathtt{u}_i$ appearing in ϕ **do**
      $U_i \leftarrow$ set of users that verify every atomic predicates $p(\mathtt{u}_i)$ in ϕ
    for all resource variable $\mathtt{r}_i$ appearing in ϕ not appearing in any *Allows* **do**
      $R_i \leftarrow$ set of resources that verify every atomic predicates $p(\mathtt{r}_i)$ in ϕ
    for all users $s \in \mathcal{U}$ that verify all the predicates over `Subject` in ϕ and
     resources $r \in \mathcal{R}_{\uparrow u} \cap \mathcal{R}_{\downarrow s}$ that verify all the predicates over `Resource` in ϕ **do**
      $\phi'_{Allow} \leftarrow \phi_{Allow}\{\mathtt{Me} \mapsto u, \mathtt{Subject} \mapsto s, \mathtt{Resource} \mapsto r\}$
      for all $u_1 \in U_1 \dots u_n \in U_n$ and $r_1 \in R_1 \dots r_m \in R_m$ **do**
        $\phi'_{Allow} \leftarrow \phi'_{Allow}\{\mathtt{u}_1 \mapsto u_1 \dots \mathtt{u}_n \mapsto u_n\}$
        $\phi'_{Allow} \leftarrow \phi'_{Allow}\{\mathtt{r}_1 \mapsto r_1 \dots \mathtt{r}_m \mapsto r_m\}$
        for all resource variables $\mathtt{r}'_i$ such that
         $Allows(\mathtt{u}_s, \mathtt{r}'_i, \mathtt{u}_o)$ occurs in ϕ for some $\mathtt{u}_s$ and $\mathtt{u}_o$ **do**
          $R_{i'} \leftarrow$ subset of $\mathcal{R}_{\uparrow u_o} \cap \mathcal{R}_{\downarrow u_s}$ that verify the predicates $p(\mathtt{r}'_i)$ in ϕ
        for all $r'_1 \in R'_1 \dots r'_k \in R'_k$ **do**
          $\phi'_{Allow} \leftarrow \phi'_{Allow}\{\mathtt{r}'_1 \mapsto r'_1 \dots \mathtt{r}'_k \mapsto r'_k\}$
          $\Gamma \leftarrow \Gamma \cup (\phi'_{Allow} \twoheadrightarrow Allow(s, r, u))$
  return Γ

Algorithm 2

for all users $u \in \mathcal{U}$ **do**
  $\sigma(u) \leftarrow$ RECEIVE-FROM(u)
$\Gamma \leftarrow$ TRANSLATE-INTO-PCL(σ) ▷ through Algorithm 1
while true **do**
  $\mathtt{asks}(s, r) \leftarrow$ RECEIVE-FROM-ANY($s \in \mathcal{U}$)
  $u \leftarrow$ OWNER(r)
  check $\leftarrow$ PCLPROVER(Γ, $Allows(s, r, u)$) ▷ checks if $\Gamma \vdash Allows(s, r, u)$
  if check **then** SEND-TO(s, *Grant*)
  else SEND-TO(s, *Deny*)

they own. In this environment, a main issue is the combination of individual policies. To the best of our knowledge, no proposals address mutuality, but only focus on the resolution of conflicts [5, 6, 9]. In the restricted, yet widespread distributed world of social networks, mutuality plays a prominent role, but is scarcely regulated. A remarkable exception is [12] that permits defining mutual access control policies. This is done by introducing a new grant, called *mutual*, besides the usual *accept* and *deny*. Suppose that an access request from user A to resource r of B evaluates to *mutual*. Intuitively, the request is served if and only if a request from B for a similar resource r' of A will evaluate to *accept* or *mutual*. Similarity is fixed once and for all, and is not user-defined. A first difference with our proposal is that mutuality is defined through explicit constraints in the body of the rules, so allowing users to define their

own notion of similarity. In addition, mutuality in *MuAC* may involve many users, as in the Example (5).

Mutuality plays a main role also in trust negotiation, a process that permits a safe interaction between two parties that do not trust each other [7]. The idea is to run a multi-round protocol where the parties exchange some pieces of private information (*credentials*) so as to increase their mutual trust. Also in this setting, each party defines an individual policy specifying the conditions the other party must satisfy in turn to obtain credentials. The overall goal is to balance the disclosure of information and the mutual benefit gained by each party. Logical languages for specifying trust policies have been proposed, e.g. Cassandra [4] and SecPal4P [3]. However, the main difference with the respect to ours is that these are based on classical logic, and thus circular conditions do not lead to an agreement.

References

[1] Pcl. http://www.disi.unitn.it/~zunino/PCL.

[2] Massimo Bartoletti and Roberto Zunino. A calculus of contracting processes. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010*, pages 332–341. IEEE Computer Society, 2010.

[3] Moritz Y. Becker, Alexander Malkis, and Laurent Bussard. A framework for privacy preferences and data-handling policies. Technical Report MSR–TR–2009–128, Microsoft Research, September 2009.

[4] Moritz Y. Becker and Peter Sewell. Cassandra: Distributed access control policies with tunable expressiveness. In *5th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2004)*, pages 159–168. IEEE Computer Society, 2004.

[5] Glenn Bruns and Michael Huth. Access control via belnap logic: Intuitive, expressive, and analyzable policy composition. *ACM Trans. Inf. Syst. Secur.*, 14(1):9:1–9:27, June 2011.

[6] Stan Damen, Jerry den Hartog, and Nicola Zannone. Collac: Collaborative access control. In *2014 International Conference on Collaboration Technologies and Systems (CTS)*, pages 142–149, May 2014.

[7] Martin Kolár, M. Carmen Fernández Gago, and Javier López. Policy languages and their suitability for trust negotiation. In Florian Kerschbaum and Stefano Paraboschi, editors, *Data and Applications Security and Privacy XXXII - 32nd Annual IFIP WG 11.3 Conference,Proceedings*, volume 10980 of *LNCS*, pages 69–84. Springer, 2018.

[8] Hal Lochhart, Bill Parducci, and Rich Levinson. OASIS eXtensible Access Control Markup Language (XACML) Version 3.0. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml, 2019. Online; last access Dec 2019.

[9] Federica Paci, Anna Cinzia Squicciarini, and Nicola Zannone. Survey on access control for community-centered collaborative systems. *ACM Comput. Surv.*, 51(1):6:1–6:38, 2018.

[10] William Stallings and Lawrie Brown. *Computer Security: Principles and Practice*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2014.

[11] Vivy Suhendra. A survey on access control deployment. In Tai-hoon Kim, Hojjat Adeli, Wai-chi Fang, Javier García Villalba, Kirk P. Arnett, and Muhammad Khurram Khan, editors, *Security Technology*, pages 11–20, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[12] Gabriela Suntaxi, Aboubakr Achraf El Ghazi, and Klemens Böhm. Mutual authorizations: Semantics and integration issues. In *Proceedings of the 24th ACM Symposium on Access Control Models and Technologies*, SACMAT '19, pages 213–218, New York, NY, USA, 2019. ACM.

[13] The Netfilter Project. Netfilter. https://www.netfilter.org/, 2019. Online; last access Dec 2019.

[14] Yunpeng Zhang and Xuqing Wu. Access control in internet of things: A survey, 2016.