Collaborative Access Decisions: Why has my decision not been enforced?

Jerry den Hartog and Nicola Zannone

Eindhoven University of Technology
{j.d.hartog,n.zannone}@tue.nl

Abstract. With the increasing popularity of collaborative systems like social networks, the risk of data misuse has become even more critical for users. As a consequence, there is a growing demand for solutions to properly protect data created and used within these systems. Enabling collaborative specification of permissions, while ensuring an appropriate levels of control to the different parties involved, inherently leads to decisions of some users being overruled by the policies of other users. Users need to be aware that this is happening and why, otherwise they may lose trust in the system, which can impact their willingness to collaborate. Enhancing user awareness requires that users know about and understand the conflicts that occurred. In this paper, we propose an approach to compute a justification for a decision in cases where conflicts occur and, based on this, generate feedback that explains users why their decision was not enforced.

1 Introduction

Recent years have witnessed an increasing popularity of collaborative systems like social networks and shared editing platforms. These systems provide virtual worlds in which their users can interact with each other and share information. Within these virtual worlds, multiple users can be involved in the creation and management of data, each of them retaining some level of authority over the data. This has spurred the design of solutions for enabling collaborative specification of permissions in which each user can specify its own authorization requirements for the protection of the data under its control [3, 12, 13, 26]. In particular, these solutions aim to ensure an appropriate levels of control to the different parties involved.

Every user expects its authorization requirements to be enforced by the system. However, this is not always possible as users can specify conflicting authorization requirements for the same resources. Most access control mechanisms employ policy conflict resolution strategies [15, 19, 21–23] to automatically determine how policy conflicts should be resolved based, for instance, on priorities between decisions (e.g., permit-overrides) or the ordering of policies (e.g., first-applicable). Although their use is necessary to guarantee the proper functioning of the system, these strategies make policy evaluation non-transparent to users. In fact, access control mechanisms usually adopt a black-box approach whose aim is only to obtain a conclusive decision to be enforced. This black-box approach results in users not being aware whether their policies have actually been enforced. The lack of transparency in decision making can effect users' experience and, consequently, their confidence in the system.

A few proposals [13, 20] make a first step towards the design of transparent access control mechanisms. In particular, Mahmudlu et al. [20] propose a feedback mechanism that identifies mismatches between the decision enforced by the system and user policies and notifies users about them. Although this feedback enhances user awareness about access decision making, it does not allow users to understand why their policies have not been enforced. Without this knowledge, users can feel that their data are not adequately protected and, thus, have a low confidence in the system. (Security and protection of private data are important factors for trust, especially for knowledgeable users [4, 5]).

In this work, we make a step further towards the design of transparent access control mechanisms by presenting an approach that not only notifies users about policy conflicts but also provides them with a meaningful explanation of why their decision has been overruled. The approach relies on the data governance model presented in [20] to represent how the authorization requirements of the users contributing to the creation and management of a data object are combined to form a global policy, which is ultimately used to regulate the access to the object. Based on the evaluation of the global policy, we identify the user policies that were used to obtain the decision enforced by the authorization mechanism, providing a justification for the decision.

Policies and decision preferences of users, however, can be sensitive themselves [27, 29]. Thus, not all users are supposed to see the full explanation of a decision. Instead, the feedback should give an appropriate level of detail, which takes into account the relationship of users with the data as well as the visibility preferences of the policy authors. To this end, we trim the explanation for a decision based on visibility restrictions, indicating which portion of the explanation a user is allowed to see. It is of utmost importance that the feedback is understandable by users. Therefore, we show how the feedback can be formulated in a human readable format, focusing on the relevant parts and customizing the feedback to reflect the relationship of the user with the data.

The remainder of the paper is organized as follows. The next section provides background on data governance and policy mismatch. Section 3 illustrates the problem of transparency in access control through a typical scenario in social network. Section 4 presents our approach to compute feedback concerning policy and to express it in a way that is understandable by end-users. Section 5 discusses related work. Finally, Section 6 concludes the paper and presents directions for future work.

2 Background

This section provides background on data governance and policy mismatch.

2.1 Data Governance Model

In collaborative systems, several users can contribute to the creation, governance and management of data. Each user can retain some authority on the data. In this work, we adopt the data governance model proposed in [20] to represent and reason on the governance of data controlled by multiple users. This model poses its basis on the notion of *archetype* [6], which is used to capture the relations of users with data objects, and uses an archetype hierarchy to represent and reason on the level of authority that users have over the data based on their archetype. An archetype hierarchy is defined as follows:

Policy combination algorithm	Source
permit-overrides (pov)	[15, 22, 23]
deny-overrides (dov)	[15, 22, 23, 25]
ordered-permit-overrides (opov)	[22, 23]
ordered-deny-overrides (odov)	[22, 23]
first-applicable (fa)	[1, 22, 23, 25]
only-one-applicable (ooa)	[22, 23]
permit-unless-deny (pud)	[23]
deny-unless-permit (dup)	[23]
specificy-precedence (sp)	[15, 21, 24, 25]
weak-consensus (wc)	[19]
strong-consensus (sc)	[13, 19]
weak-majority (wm)	[19]
strong-majority (sm)	[13, 19]
super-majority-permit (smp)	[13, 19]

Table 1: Policy Combining Algorithms for XACML

Definition 1. *Let $\mathcal{A}$ be the set of archetypes for a data object o. An* archetype hierarchy *H has the form:*

$$\begin{aligned} H &= SH \mid (SH, t, H) \\ SH &= L \mid (L, \oplus, SH) \mid (L, \ominus, SH) \\ L &= a \mid (\sigma[a_1, \ldots, a_n]) \end{aligned}$$

An archetype hierarchy H is (recursively) built over sub-hierarchies (SH) and levels (L) by concatenating them according to a given priority that can be total (denoted by t), positive (denoted by $\oplus$) or negative (denoted by $\ominus$). A level L consists of an archetype a or a set of archetypes $a_1, \ldots, a_n \in \mathcal{A}$ that are combined using intra-level aggregator σ.

An archetype hierarchy is used to combine stakeholders' authorization requirements into a *global policy*, which regulates the access to data. In particular, the work in [20] supports the definition of the global policy for a data object from stakeholders' authorization requirements specified as XACML policies (hereafter called *user policies*). The combination of user policies in the global policy reflects the level of authority that stakeholders have over the object as defined in the archetype hierarchy. The underlying idea is to represent priorities between levels and intra-level aggregators as policy combining algorithms. Here, we do not impose any restriction on the combining algorithms that can be used. The only requirement is that they can be implemented in XACML. Table 1 presents an overview of policy combining algorithms that have been proposed for and/or adapted to XACML.[1]

For the sake of simplicity, in this work we abstract from the XACML specification (e.g., target, rule, policy, policy set), while keeping full compatibility with the standard.

[1] We assume the reader is familiar with conflict resolution strategies and, in particular, with XACML policy combining algorithms.

We represent (XACML) policies as trees where nodes are labeled with a combining algorithm and leaf nodes are labeled with user policies. In particular, we represent policy trees either in graphical (see e.g. Figure 1b) or textual form where $ca(\Delta_1, \ldots, \Delta_n)$ represents a node labeled with combining algorithm ca and subtrees $\Delta_1, \ldots, \Delta_n$.

It is worth noting that our representation of policies accounts for user policies as atomic elements regardless of whether they are composite policies themselves. This is due to the fact that the feedback mechanism proposed in this work focuses on the governance of data controlled by multiple users and, in particular, aims to identify the users whose policies have overridden the policy of a given user. Therefore, this level of granularity is adequate for our scope.

Below we present how the global policy is constructed from the archetype hierarchy and user policies.

Definition 2. *Given a data object o, let $\mathcal{A}$ be the set of archetypes for o, H the archetype hierarchy built over $\mathcal{A}$, $\mathcal{U}$ the set of user identifiers (or simply users) and $\mathcal{P}_\mathcal{U}$ the set of user policies where $p_u \in \mathcal{P}_\mathcal{U}$ denotes the policy of user $u \in \mathcal{U}$. Let $UA \subseteq \mathcal{U} \times \mathcal{A}$ be the user-archetype assignment, i.e. $(u, a) \in UA$ iff user u has archetype a. We construct the* global policy P_H *for H starting from the top of H:*

$$\begin{aligned} P_{(SH,t,H)} &= \mathsf{fa}(P_{SH}, P_H) \\ P_{(L,\oplus,SH)} &= \mathsf{opov}(P_L, P_{SH}) \\ P_{(L,\ominus,SH)} &= \mathsf{odov}(P_L, P_{SH}) \\ P_{(\sigma,[a_1,\ldots,a_n])} &= ca_\sigma(P_{a_1}, \ldots, P_{a_n}) \\ P_a &= ca_a(p_{u_1}, \ldots, p_{u_m}) \end{aligned}$$

where ca_σ is the combining algorithm realizing the intra-level aggregator σ, ca_a the combining algorithm associated with archetype $a \in \mathcal{A}$ and $p_{u_1}, \ldots, p_{u_m} \in \mathcal{P}_\mathcal{U}$ where $u_1, \ldots, u_m$ are the users such that $(u_1, a), \ldots, (u_m, a)$ are in UA.

For some objects and archetypes it is natural that there is only a single user associated to a given archetype. In this case we use only-one-applicable as archetype combining algorithm ca_a. This way the decision of the (only) user policy becomes the decision of the archetype and the presence of multiple decisions would result in an error (Indeterminate).

The global policy for a data object is used to determine whether access to the object should be granted or not. We use the following abstract notation to represent the policy evaluation process: $\mathcal{P}$ denotes the set of XACML policies, $\mathcal{Q}$ the set of access requests, and function $[\![p]\!] : \mathcal{Q} \to \{\mathsf{Permit}, \mathsf{Deny}, \mathsf{NotApplicable}, \mathsf{Indeterminate}\}$ denotes policy evaluation, i.e. $[\![p]\!](q)$ is the decision according to a policy $p \in \mathcal{P}$ for an access request $q \in \mathcal{Q}$. In particular, Permit (P) denotes that access is granted, Deny (D) denotes that access is denied, NotApplicable (NA) denotes that the policy is not applicable, and Indeterminate (I) denotes that an error occurred during evaluation.

2.2 Policy mismatch

Ideally, an authorization mechanism should enforce the authorization requirements of all users. However, this is not always possible. In fact, users can specify conflicting authorization requirements, which results in conflicting policies. In this work, we use

the notion of *policy mismatch* introduced in [6, 20] to capture that the decision yielded by a user policy differs from the one obtained by evaluating the global policy.

Definition 3. *Let $p_1, \ldots, p_n$ be the policies of n users and p the global policy obtained by combining such policies. Given an access request q, a user u (with $u \in \{1, \ldots, n\}$) has a* policy mismatch *if $[\![p_u]\!](q) \neq [\![p]\!](q)$.*

The notion of policy mismatch provides the baseline for enabling transparency in access control. For instance, Mahmudlu et al. [20] show how to augment SAFAX [16], an XACML-based architectural framework that offers authorization as a service, with a transparency service that detects mismatches between the decision enforced by the authorization mechanism and users' authorization requirements. Any mismatch found is reported to those users whose decision was not enforced.

3 Motivating Example

This section illustrates the motivation for this work using a FaceBook-like social network augmented with a collaborative access control system in the style of [20].

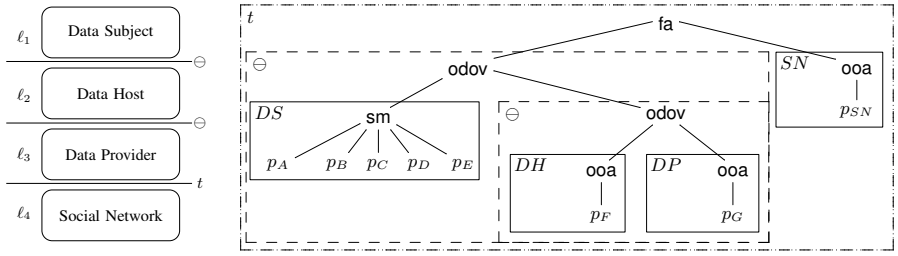
Example 1. An online social network provides a collaborative environment in which users can post messages and photos in their profile and share these objects with other users. Users can also post messages and photos in the profile of other users (if they have permission) and tag a data object to indicate the user(s) to whom the object refers.

To regulate the access to data, the social network allows users to specify their privacy settings. A user's privacy settings govern the actions that users (or groups of users, e.g. Friends, Colleagues) in the social network can perform on the objects (profile, posts, etc.) controlled by the user. The social network also defines a default policy that is used to handle the situations in which users do not specify their privacy settings.

Our scenario focuses on a user who posts a photo in the profile of another user. The photo shows five individuals, who are registered to the social network. These users are tagged and, thus, the photo is linked to their profile.

In the scenario above, we can identify four archetypes for the photo: *Data Subject* (DS), *Data Host* (DH), *Data Provider* (DP) and *Social Network* (SN). The Data Subject archetype is used to represent the individual(s) to whom the (personal) data refer. In our scenario, this archetype denotes the users appearing in the photo.[2] The Data Host archetype is used to represent the user owning the profile in which the photo has been posted. The Data Provider archetype is used to denote the user who posted the photo. Social networks usually define default settings that are used if users do not specify custom settings. Given the collaborative nature of our setting, we assume that default settings apply to the collaboration (in contrast to single users) and, thus, they are only

[2] Note that the problem of recognizing the subjects of a piece of information is orthogonal to the scope of this work. Here, we assume that tags are reliable, i.e. they link a piece of information to the corresponding data subjects. Although it is not addressed in this work, tag validation has been proven to be feasible and, for instance, several algorithms have been proposed to automatically recognize people in contents such as photos [13].



(a) Archetype hierarchy (b) Global policy

Fig. 1: Data Governance Model and Instantiation

considered if no other settings have been specified by any user. We capture these default settings within the governance of the photo through the Social Network archetype.

The identified archetypes can be organized in a hierarchy (Fig. 1a). We assume that the Data Subject has the highest priority as it should be able to influence the processing of its personal data [11]. The next level comprises the Data Host, who is responsible for the contents posted in its profile, followed by a level formed by the Data Provider. The lowest level is formed by the Social Network. The first three levels are ordered using a negative priority ($\ominus$), meaning that the negative authorization requirements (i.e., requirements explicitly denying access to data) associated to the higher level take precedence; otherwise, the access requirements defined by the stakeholders at the lower level should also be evaluated. The default settings defined by the Social Network is overridden by the settings of the other stakeholders. We capture this requirement using a total priority (t) between the Social Network and higher levels.

The global policy p is obtained by instantiating the archetype hierarchy in Fig. 1a with user policies. Let users A, B, C, D and E be the Data Subjects (i.e., the users appearing in the photo), user F the Data Host and user G the Data Provider. Each of these users can define a (possibly empty) policy to regulate the access to their data. Moreover, we use p_{SN} to denote the default settings provided by the social network. Textually, the global policy can be represented as follows:

$$p = \mathsf{fa}(\mathsf{odov}(\mathsf{sm}(p_A, p_B, p_C, p_D, p_E), \mathsf{odov}(\mathsf{ooa}(p_F), \mathsf{ooa}(p_G))), \mathsf{ooa}(p_{SN}))$$

A graphical representation of the global policy as a policy tree is shown in Fig. 1b. Priorities in the archetype hierarchy are encoded as combining algorithms in the global policy as defined in Definition 2. Levels and archetypes are defined along with a combining algorithm that specifies how archetype policies and user policies forming them should be combined respectively. Here, we assume that the policies specified by data subjects are combined using the strong-majority (sm) combining algorithm proposed in [19]. According to this combining algorithm, access is granted if over half of all subpolicies allow it, and deny access if over half deny it; otherwise, an indeterminate decision is returned. The other archetypes (i.e., Data Host, Data Provider and Social Network) are associated to only one user. As described in Section 2, we use only-one-applicable (ooa) as the archetype combining algorithm for these archetypes. Similarly, all levels consist of only one archetype. Accordingly, they are represented as the archetype forming them (see Definition 1).

Example 1 (Cont.). Suppose a user u requests to view the photo. The authorization system has to evaluate the access request q made by u against the global policy p in Fig. 1b. Assume user policies are evaluated as follows:

$$\begin{array}{ll} [\![p_A]\!](q) = \mathsf{D} & [\![p_E]\!](q) = \mathsf{D} \\ [\![p_B]\!](q) = \mathsf{D} & [\![p_F]\!](q) = \mathsf{NA} \\ [\![p_C]\!](q) = \mathsf{P} & [\![p_G]\!](q) = \mathsf{P} \\ [\![p_D]\!](q) = \mathsf{D} & [\![p_{SN}]\!](q) = \mathsf{P} \end{array}$$

Accordingly, the request is denied by the authorization mechanism, i.e. $[\![p]\!](q) = \mathsf{D}$.

We can observe that the authorization requirements of some users have not been enforced. For instance, the requirements of users C and G allows the requester to view the photo. The default policy p_{SN} has also been overridden, indicating that it may be too permissive for certain users. Moreover, we can observe that some users (e.g., the data host F in our scenario) might not have specified any authorization requirement to handle certain access requests.

Every user expects its policies to be enforced by the authorization mechanism; however, as shown in the example above, the policy of some users can be overridden by the policies of other users. Although the use of strategies that automatically resolve policy conflicts is necessary to guarantee the proper functioning of the system, users are often unaware whether their policies have actually been enforced. The main problem is that most of the existing authorization mechanisms only aim to obtain a conclusive decision to be enforced and do not identify and/or record policy mismatches. We argue that this lack of transparency can affect the collaboration among users and, in particular, their willingness of sharing sensitive information.

A few works [13, 20] propose feedback mechanisms that detect and notify the user of policy conflicts. These solutions, for instance, would notify users C and G that access has been denied despite their policies granting it. Although this feedback enhances user awareness about the access decision making process, it does not allow users to understand why their policies have not been enforced. Without this knowledge, users can feel that their data are not adequately protected and, thus, have a low confidence in the system. In this work, we investigate the problem of designing *fully* transparent authorization mechanisms that are able to explain to users why a certain access decision has been made.

Although it is crucial that users understand why their policies have been overridden, the feedback generation should be separated from policy evaluation. Certain systems like critical infrastructures require a fast response time and, thus, any delay introduced by the feedback generation could compromise the functioning of the system. To achieve this separation of concerns, we envision transparency as a service. Similarly to [20], we decouple the feedback mechanism from policy evaluation, thus relieving the burden of computing the user feedback from the policy evaluation engine. This design choice has the added benefit that authorization mechanisms already in place can easily be augmented with transparency, thus facilitating the adoption of transparency in existing systems. In the next section, we present a framework with a feedback mechanisms that not only notify users if a policy mismatch occurred but also provide them with a justification of why their policies have not been enforced.

4 Approach

Upon receiving an access request, the authorization mechanism evaluates the request against the global policy to determine the access decision to be enforced. However, as shown in the previous section, the authorization requirements of some users might have to be overridden in order for the authorization mechanism to reach a conclusive decision. The goal of this work is to raise awareness of users about the enforcement of their authorization requirements. This section presents our approach to generating feedback which explains to users why their authorization requirements have not been enforced. The approach, shown in Figure 2, consists of four main steps.

The first step is to find *policy mismatches*, i.e. those situations in which user policies have been overridden (see Definition 3). To detect policy mismatches, we employ the transparency service presented in [20]. This service identifies mismatches and users involved by comparing the decision for a request according to the global policy with the decision according to user policies evaluated individually. The service also provides a feedback mechanism for mismatches which notifies the users involved. We refer to [20] for details on the transparency service for policy mismatch detection and notification.

Although this transparency service makes users aware of whether or not their policies have been enforced, notifications should be extended to provide an explanation of why a user's policy was overridden in order to increase user awareness in access decision making. To provide such explanations, we compute the *decision annotated evaluation path*, which provides a justification for the decision enforced by the system (step 2). Intuitively, a decision annotated evaluation path comprises a (minimal) set of user policies (along with their evaluation) that allows the system to show why a certain decision was obtained.

A decision annotated evaluation path provides a "technical" explanation of why a certain decision was reached. End-users know the archetype hierarchy but may not be able to interpret explanations based on the global policy. Therefore, we express feedback in terms of the archetype hierarchy, to give users the information needed to understand the decision justification. Also, a decision annotated evaluation path may reveal information about the policies of other users. Policies themselves can be sensitive [27, 29] and, thus, need to be protected. To this end, we employ *visibility policies* to regulate the information disclosed in the feedback (step 3). In particular, visibility policies are used to determine *visibility restrictions* on the justification, indicating which portion of the justification should be visible to a user based on its place in the archetype hierarchy, and to *trim* the justification accordingly. In this work we assume that users set the visibility policies of their own access control policies, whereas the visibility policies of the other elements (e.g., archetypes, levels) are defined during the setting of the collaboration along with the archetype hierarchy.

Note that we have separated the computation of the justifications for a decision from the computation of the feedback. An advantage of this separation is that the feedback can be customized with respect to the relation of the user to be notified with the data object. In particular, the granularity of the feedback given to end-users can be tuned on the basis of the needs of the application domain and visibility restrictions without modifying the procedure used to compute the feedback.

It is important that the feedback is understandable by the users. In addition to relating it to terms they know (the archetype hierarchy) we show how the feedback can

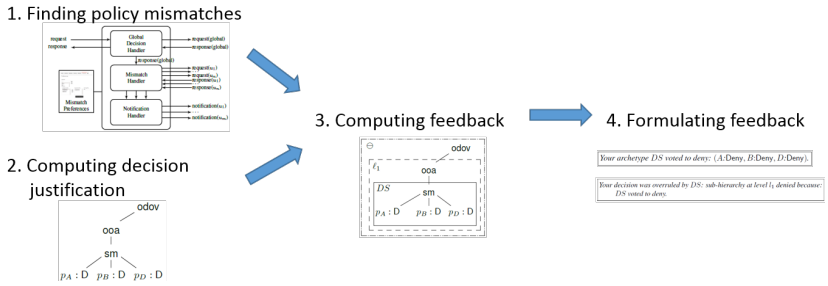


Fig. 2: Approach to enhance user awareness in access decision making

be formulated into a human readable format (step 4). In particular, we transform the justification for a decision trimmed with respect to visibility restrictions into a textual description, focusing on the relevant parts and customizing the feedback to reflects the user's place in the archetype hierarchy.

4.1 Computing Decision Justifications

As seen above, we use an (ordered) labeled tree to represent the global policy where nodes are labeled with a combining algorithm and leaf nodes with user policies. Formally, an *ordered tree* is a set of nodes $\mathcal{N}$ with a partial order amongst the nodes and a total order amongst the children of each node. A labeling of a tree is a function from nodes to some domain of labels. A *labeled tree* is a tree with one or more labellings. Recall that we use $ca(\Delta_1, \ldots, \Delta_m)$ to indicate a node labeled with a combining algorithm ca and subtrees $\Delta_1, \ldots, \Delta_m$. Note that this notation defines both the tree structure and a labeling. Moreover, we refer to a connected subgraph of a tree containing the root as a *pruning* of the tree. Note that a pruning is itself a tree and the union of multiple prunings is again a pruning.

We introduce an additional label to the global policy in order to capture decisions reached.

Definition 4. *Let $\mathcal{N}$ be the set of nodes in the global policy and $\mathcal{Q}$ the set of access requests. The* Decision *labeling with respect to an access request $q \in \mathcal{Q}$ is a labeling $D_q : \mathcal{N} \to \{\mathsf{Permit}, \mathsf{Deny}, \mathsf{NotApplicable}, \mathsf{Indeterminate}\}$. A node $n \in \mathcal{N}$ is labeled with a decision according to the policy that the subtree of n represents:*

- *for n labeled with user policy p, $D_q(n)$ is $[\![p]\!](q)$;*
- *for n labeled with combining algorithm ca, $D_q(n)$ is the result of ca applied to decision list $D_q(n_1), \ldots, D_q(n_m)$ where $n_1, \ldots, n_m$ are the children of n.*

The *Decision* labeling denotes the outcome of policy evaluation with respect to a given access request. For nodes labeled with a combining algorithm, the Decision label is the result of applying that combining algorithm to the decision labels of its children. Note that this is equivalent to evaluating the policy tree rooted in n, i.e. $D(n) = [\![ca(\Delta_1, \ldots, \Delta_m)]\!](q)$ with $\Delta_1, \ldots, \Delta_m$ the subtrees of n.

Example 2. The Decision labeling of the global policy in Figure 1b, labeled according to the decisions of user policies as given in Example 1 (page 7), is:

$$\mathsf{fa{:}D}(\mathsf{odov{:}D}(\mathsf{sm{:}D}(p_A{:}\mathsf{D}, p_B{:}\mathsf{D}, p_C{:}\mathsf{P}, p_D{:}\mathsf{D}, p_E{:}\mathsf{D}),\\ \mathsf{odov{:}P}(\mathsf{ooa{:}NA}(p_F{:}\mathsf{NA}), \mathsf{ooa{:}P}(p_G{:}\mathsf{P}))), \mathsf{ooa{:}P}(p_{SN}{:}\mathsf{P}))$$

Note that, if only the decisions of the user policies are given, the other decisions can be computed. Thus, without loss of information, we may as well write:

$$\mathsf{fa}(\mathsf{odov}(\mathsf{sm}(p_A{:}\mathsf{D}, p_B{:}\mathsf{D}, p_C{:}\mathsf{P}, p_D{:}\mathsf{D}, p_E{:}\mathsf{D}),\\ \mathsf{odov}(\mathsf{ooa}(p_F{:}\mathsf{NA}), \mathsf{ooa}(p_G{:}\mathsf{P}))), \mathsf{ooa}(p_{SN}{:}\mathsf{P}))$$

In order to compute the feedback to be sent to a user, we first need to identify which user policies have been used to obtain a certain decision. To this end, we introduce the notion of decision annotated evaluation path.

Definition 5. *Given an access request $q \in \mathcal{Q}$, let p be the global policy with Decision labeling with respect to q. A* decision annotated evaluation path *for q is a minimal pruning of p that justifies decision $[\![p]\!](q)$.*

A decision annotated evaluation path can be seen as the set of (decision annotated) user policies that allows the system to show how a certain decision was obtained, thus representing a justification for the decision. A decision annotated evaluation path is minimal, i.e. if any node is removed, it no longer forms a justification for the decision. To prune the (decision annotated) global policy to a decision annotated evaluation path we can start from the root and recursively, for each node labeled with a combining algorithm ca, only include a minimal subset of children that justify the decision label (according to ca). Note that the pruning depends on the semantics of the combining algorithms with respect to a given decision. For the sake of space, we omit the formal definition of minimal pruning and only provide the intuition for a few algorithms.

deny-overrides returns Deny if and only if one of the subpolicies returns Deny. Therefore, to show that a policy $\mathsf{dov}(\Delta_1, \ldots, \Delta_m)$ is evaluated Deny, it is sufficient to show that one of the subpolicies evaluate Deny. In contrast, for the other decisions (i.e., Permit, NotApplicable, Indeterminate) the decision annotated evaluation path should provide all (decision annotated) subpolicies as the system has to show that none of the subpolicies evaluate Deny.

ordered-deny-overrides is identical to deny-overrides with the exception that subpolicies are considered in the order in which they are defined. Accordingly, the decision annotated evaluation path will contain the first subpolicy that evaluates Deny if any; otherwise, if no subpolicies evaluate Deny, all (decision annotated) subpolicies are included in the decision annotated evaluation path.

first-applicable returns the decision of the first applicable policy. Accordingly, the decision annotated evaluation path contains the policy used to make the decision together with the previous policies. In fact, the system should show that none of these previous policies is applicable. Following the same intuition, if none of the subpolicies are applicable, all subpolicies are given in the decision annotated evaluation path.

strong-majority returns a conclusive decision, either Permit or Deny, if over half of all subpolicies evaluate Permit and Deny respectively; otherwise, Indeterminate is

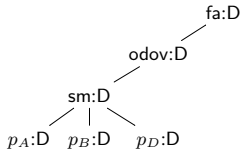


Fig. 3: Decision annotated evaluation path

returned. Accordingly, it is sufficient for the system to only show that over half of all subpolicies return Permit (Deny) to prove that the policy is evaluated Permit (Deny). On the other hand, in case of Indeterminate, the system has to show that a majority of either Permit or Deny cannot be reached.

Example 3. Based on the evaluation of Example 1 (page 7), the requester is not allowed to view the photo. Analyzing the global policy, we can observe that access is denied because the majority of data subjects deny the access. In particular, four data subjects out of five stated in their policies that access should be denied. To show why a Deny decision was reached, the system has only to show that three data subjects denied the access (i.e., the majority). Accordingly, the following decision annotated evaluation path justifies the decision:

$\mathsf{fa}(\mathsf{odov}(\mathsf{sm}(p_A{:}\mathsf{D}, p_B{:}\mathsf{D}, p_D{:}\mathsf{D})))$

Fig. 3 shows a graphical representation of this decision annotated evaluation path. It is easy to observe from the figure that it is a minimal pruning of the global policy in Fig. 2 that justifies the decision obtained.

4.2 Computing Feedback

In this section, we show how a decision annotated evaluation path can be used to compute the feedback. First, we present how to link the global policy to the archetype hierarchy. Then, we propose an approach to determine the granularity at which a user can see the feedback based on its place in the archetype hierarchy.

Linking the global policy to the archetype hierarchy A decision annotated evaluation path represents how the decision for a given access request has been reached. However, it only provides a purely "technical" explanation in terms of partial decisions and combining algorithms. Feedback based on the archetype hierarchy rather than on the details of its implementation will likely be easier to understand by end-users.

To enable such a feedback we relate the global policy, and thus also any decision annotated evaluation path, to the archetype hierarchy by introducing an additional labeling of the global policy.

Definition 6. *Let $\mathcal{N}$ be the set of nodes in the global policy, $\mathcal{H}$ the set of elements in the archetype hierarchy H (i.e., priorities, levels, archetypes) from which the global policy is derived and $\mathcal{U}$ the set of user identifiers. The* Element *labeling is a labeling $E : \mathcal{N} \to \mathcal{H} \cup \mathcal{U}$. A node $n \in \mathcal{N}$ is labeled as follows:*

- *For n labeled with a combining algorithm modeling hierarchy element (SH, t, H), $E(n)$ is t;*
- *For n labeled with a combining algorithm modeling hierarchy element $(L, \oplus, SH)$, $E(n)$ is $\oplus$;*
- *For n labeled with a combining algorithm modeling hierarchy element $(L, \ominus, SH)$, $E(n)$ is $\ominus$;*
- *For n labeled with a combining algorithm modeling hierarchy element $(\sigma[a_1, \ldots, a_n])$, $E(n)$ is ℓ where ℓ is the level identifier;*
- *For n labeled with a combining algorithm modeling an archetype a, $E(n)$ is a;*
- *For n labeled with a policy p of user u, $E(n)$ is u.*

Labeling E annotates the global policy with the corresponding element in the hierarchy and with the users contributing to its definition. Hereafter, we use notation $[\cdot]$ to represent Element labels, e.g. $p[e]{:}d$ denotes a node with policy element label p, hierarchy element label e and decision label d.

The Element labeling provides us with the complete information about the construction of the global policy, which is needed to provide users meaningful feedback and to compute the visibility of the feedback as shown in the remainder of the section.

Example 4. The decision annotated evaluation path of Example 3, annotated with *Element* labeling, is: $\mathsf{fa}[t]{:}\mathsf{D}(\mathsf{odov}[\ominus]{:}\mathsf{D}(\mathsf{sm}[DS]{:}\mathsf{D}(p_A[A]{:}\mathsf{D}, p_B[B]{:}\mathsf{D}, p_D[D]{:}\mathsf{D})))$
(or in short notation: $\mathsf{fa}[t](\mathsf{odov}[\ominus](\mathsf{sm}[DS](p_A[A]{:}\mathsf{D}, p_B[B]{:}\mathsf{D}, p_D[D]{:}\mathsf{D}))))$

Reasoning on Feedback Visibility One can observe in Example 4 that the justification for a decision can provide insights into the policies of other users. While it may be reasonable for the collaborating data subjects (e.g., C in our scenario) to see the individual votes of the fellow data subjects, they may wish to not reveal this information to other users (e.g., to the data provider G). Policies and decision preferences of users might be sensitive; thus, not all users are supposed to see the full explanation of a decision. Yet, they do need to get informative feedback if their policies have not been enforced.

To determine at which granularity a user can see the justification for a decision, we annotate the global policy with a visibility policy, indicating which portion of the decision annotated evaluation path is visible to users based on their place in the archetype hierarchy, and show how the visibility policy can be used to trim the justification. Visibility policies are expressed in terms of visibility levels.

Definition 7. *The* visibility classification *is a pair $(\mathcal{V}, >)$ where $\mathcal{V} = \{User, Archetype, Level, Subhierarchy, Hierarchy, Decision\}$ is the set of visibility levels and $>$ is a total order on $\mathcal{V}$ such that:*

$$User > Archetype > Level > Subhierarchy > Hierarchy > Decision$$

We extend this to $\mathcal{V}_\perp$ by adding $\perp$ (undefined) which is smaller than any level. Given two visibility levels v_i and v_j, we use $v_i \wedge v_j$ to denote the minimum and $v_i \vee v_j$ to denote the maximum visibility level between v_i and v_j with respect to $>$, i.e.

$$v_i \wedge v_j = \begin{cases} v_j & \text{if } v_i > v_j \\ v_i & \text{otherwise} \end{cases} \qquad v_i \vee v_j = \begin{cases} v_i & \text{if } v_i > v_j \\ v_j & \text{otherwise} \end{cases}$$

Moreover, $v_i \triangleright v_j$ *denotes that* v_i *overrides* v_j*, i.e.*

$$v_i \triangleright v_j = \begin{cases} v_i \text{ if } v_i \neq \bot \\ v_j \text{ otherwise} \end{cases}$$

Visibility levels define the granularity at which justifications can be seen in terms of types of nodes. The finest level is $User$ which allows seeing decisions of users. $Archetype$ instead only allows seeing the decision reached by the archetype but not the users within the archetype (e.g., for DS we see the results of the 'vote' but not any of the votes themselves). $Level$ abstracts a step further allowing only the level decision to be seen (in our example each level consists of only one archetype so this is not a mayor distinction, but in general a level may consist of multiple archetypes [20]). $Subhierachy$ allows seeing the decision of the sub-hierarchies but not the levels themselves. $Hierarchy$ abstracts a step further, allowing only seeing the decision of total priorities. $Decision$ only allows seeing the end result and not how this decision was reached.

In general, not all users will be allowed to see a justification at the same granularity. Instead, just like the access rights, 'visibility' rights depend on the relation users have to the object considered. As such we assign an internal and external visibility level to the different components of the hierarchy and combine these to reach a visibility level for each mismatch that occurs.

Definition 8. *Let* $\mathcal{N}$ *be the set of nodes in the global policy and* $(\mathcal{V}, >)$ *the visibility classification. A* visibility policy *is a labeling* $V : \mathcal{N} \to \mathcal{V} \times \mathcal{V}$*. For a node* $n \in \mathcal{N}$*, the visibility policy label* $V(n)$ *is a pair* $\langle e, i \rangle$ *where* e *is the external visibility level and* i *is the internal visibility level of* n*.*

A visibility policy determines whether only the decision or some detail of the internal decision making structure is visible. Setting the visibility level of a node lower than the type of the node it is assigned to, means only the decision is visible. Setting a higher level allows some visibility of the decision making structure but only up to the level given and only so far as the subtree allows it. Setting the visibility level of a node equal to the type of the node will also result in showing only the decision (as all subtrees will have a higher visibility level so will not be visible) with the exception of priority nodes that can have other priority nodes as a child node.

With a visibility policy in place we can determine which part of a justification a user can see. Recall that a justification is an (annotated) pruning of the global policy. The underlying idea for determining which nodes of the justification are visible to a given user, is to take the shortest path from the user to the node (i.e., up to the least upper bound and then down to the node) and take the minimum visibility level encountered, where in each step the internal visibility of the destination is used when moving up and its external visibility when moving down. When this minimum visibility is greater or equal to the type of the node, then the node is visible to the user.

We formalize this process in two steps. First, we compute the *visibility restriction* of the nodes in the global policy by moving up through the policy tree. Then, we use the computed visibility restriction to *trim* a justification by moving down through the policy tree. The visibility restriction captures the location of a user compared to nodes by combining internal policies that apply to the user for that node.

Definition 9. *Let $\mathcal{N}$ be the set of nodes in the global policy, $\mathcal{U}$ the set of user identifiers and $\mathcal{V}$ the set of visibility levels. The* visibility restriction *with respect to a user $u \in \mathcal{U}$ is a labeling $VR_u : \mathcal{N} \to \mathcal{V}_\perp$. The visibility restriction of a user police node with respect to a given user u is:*

$$VR_u(p[u']\langle\cdot, i\rangle) = \begin{cases} i & \text{if } u = u' \\ \perp & \text{otherwise} \end{cases}$$

and if node $n\langle\cdot, i\rangle$ has children $n_1, \ldots, n_m$ then:

$$VR_u(n\langle\cdot, i\rangle) = i \wedge (\, VR_u(n_1) \vee \ldots \vee \, VR_u(n_m))$$

We extend our label notation by writing $ca[k]\langle e, i\rangle(...)|^u_x$ for a node with element label k, external policy label e, internal policy label i and restriction x with respect to user u. Moreover, we write $\Delta|^u$ to indicate a policy tree Δ with restriction labeling with respect to user u. Note that we only write the relevant labels but still assume all labellings are present.

The visibility restriction is used to trim a tree, removing those parts that should not be visible to a user.

Definition 10. *The trimming $T(\Delta)$ of a global policy tree Δ with element, visibility policy and visibility restriction (with respect a user u) labellings is given by: if $\tau(k) > x$ then $T(\Delta[k]|^u_x) = ()$, otherwise $T(p_{u'}|^u_{User}) = p_{u'}$ and*

$$T(ca(\Delta_1\langle e_1, \cdot\rangle|^u_{x_1}, \ldots, \Delta_n\langle e_n, \cdot\rangle|^u_{x_n})|^u_x) = ca(T(\Delta_1|^u_{x_1 \triangleright (x \wedge e_1)}), \ldots, T(\Delta_n|^u_{x_n \triangleright (x \wedge e_n)}))$$

where τ is a function that returns the type of a node. (Recall that visibility levels are expressed in terms of node types.)

The user is restricted from seeing a node (and its children) if the visibility is lower than the type of the node. Otherwise, the user can see the node and, if the node is labeled with a combining algorithm, we trim its subtrees but with updated visibility labels. If the user is internal to a subtree Δ_i, then its restriction (of its root) x_i is not $\perp$ and it will be used as visibility in this subtree. If the user is external to this subtree then both the current visibility restriction and the external visibility of the subtree apply which is captured by restriction $x \wedge e_i$.

Example 5. As shown in Example 1, users C, G and SN had a policy mismatch. The justification for the decision is as given in Example 4:

$\Delta = \mathsf{fa}[t](\mathsf{odov}[\ominus](\mathsf{sm}[DS](p_A[A]{:}\mathsf{D}, p_B[B]{:}\mathsf{D}, p_D[D]{:}\mathsf{D})))$

Suppose the visibility policy requirements are:

- The identity of data subjects are only visible to fellow data subjects.
- The social network can only see the end decision.

These requirements can be captured by setting external visibility of DS to $Archetype$ and internal visibility of SN to $Decision$. All other policies are set to the most liberal setting: $User$.

User C: As C is local to archetype DS, within its visibility restriction, DS will have a local visibility restriction label $User$:

$$\Delta|^C = \mathsf{fa}[t](\mathsf{odov}[\ominus](\mathsf{sm}[DS]\langle Archetype, \cdot\rangle(p_A[A]{:}\mathsf{D}, p_B[B]{:}\mathsf{D}, p_D[D]{:}\mathsf{D})|^C_{User})|^C_{User})|^C_{User}$$

The presence of this local label prevents the $\mathsf{sm}[DS]$ external policy from begin applied:

$$\begin{aligned} T(\Delta|^C) &= \mathsf{fa}[t](T(\mathsf{odov}[\ominus](\mathsf{sm}[DS]\langle Archetype, \cdot\rangle(p_A[A]{:}\mathsf{D}, p_B[B]{:}\mathsf{D}, p_D[D]{:}\mathsf{D})|^C_{User})|^C_{User})) \\ &= \mathsf{fa}[t](\mathsf{odov}[\ominus](T(\mathsf{sm}[DS](p_A[A]{:}\mathsf{D}, p_B[B]{:}\mathsf{D}, p_D[D]{:}\mathsf{D})|^C_{User \triangleright Archetype}))) \\ &= \mathsf{fa}[t](\mathsf{odov}[\ominus](T(\mathsf{sm}[DS](p_A[A]{:}\mathsf{D}, p_B[B]{:}\mathsf{D}, p_D[D]{:}\mathsf{D})|^C_{User}))) \\ &= \mathsf{fa}[t](\mathsf{odov}[\ominus](\mathsf{sm}[DS](p_A[A]{:}\mathsf{D}, p_B[B]{:}\mathsf{D}, p_D[D]{:}\mathsf{D}))) \end{aligned}$$

Therefore, C is allowed to see the entire justification.

User G: User G's visibility restriction initially allows seeing users:

$$\Delta|^G = \mathsf{fa}[t](\mathsf{odov}[\ominus](\mathsf{sm}[DS]\langle Archetype, \cdot\rangle(p_A[A]{:}\mathsf{D}, p_B[B]{:}\mathsf{D}, p_D[D]{:}\mathsf{D}))|^G_{User})|^G_{User}$$

However, being external to archetype DS causes the archetype's external policy to apply hiding the decisions of users A through E:

$$\begin{aligned} T(\Delta|^G) &= \mathsf{fa}[t](T(\mathsf{odov}[\ominus](\mathsf{sm}[DS]\langle Archetype, \cdot\rangle(p_A[A]{:}\mathsf{D}, p_B[B]{:}\mathsf{D}, p_D[D]{:}\mathsf{D}))|^G_{User})) \\ &= \mathsf{fa}[t](\mathsf{odov}[\ominus](T(\mathsf{sm}[DS](p_A[A]{:}\mathsf{D}, p_B[B]{:}\mathsf{D}, p_D[D]{:}\mathsf{D})|^G_{Archetype}))) \\ &= \mathsf{fa}[t](\mathsf{odov}[\ominus](\mathsf{sm}[DS]{:}\mathsf{D})) \end{aligned}$$

User SN: The social network has visibility restriction label $Decision$:

$$\Delta|^{SN} = \mathsf{fa}[t](\mathsf{odov}[\ominus](\mathsf{sm}[DS]\langle Archetype, \cdot\rangle(p_A[A]{:}\mathsf{D}, p_B[B]{:}\mathsf{D}, p_D[D]{:}\mathsf{D})))|^{SN}_{Decision}$$

Therefore, SN will be allowed to see only the decision and an empty explanation:

$$T(\Delta|^{SN}) = ()$$

4.3 Formulating feedback

The feedback for a user computed by the visibility restricted evaluation path technically captures the information available to that user. However, it still has to be formulated in a way that it is understandable by end-users. This requires focusing on the relevant parts and customizing the feedback to reflects its place in the archetype hierarchy in addition to translating the path into a human readable format. In fact, although formal languages are very good to provide a precise model, they are very bad at communicating such a model to end-users who might not have a technical background [18].

In our example, the policy of each data subject provides an indication whether the access should be granted to the requester while the node represented by archetype DS makes an actual decision (by counting votes, the nodes above 'simply pass up the decision'). We capture this notion of node making the decision as the *decision point*.

Definition 11. *Given a decision annotated evaluation path Δ, the* decision point *of Δ, denoted $dp(\Delta)$, is the node in Δ where the final decision is made. We call* visible decision point *for a user u the least ancestor of the decision point that occurs in $T(\Delta|^u)$.*

The decision point for a decision annotated evaluation path is recursively computed from the root node on the basis of the combining algorithms used and the decision made. We present the intuition for some combining algorithms in Table 2.

$$dp(\mathsf{dov}(\Delta_1, \ldots, \Delta_m)) = \begin{cases} dp(\Delta_i) & \text{if } \Delta_i\text{:D} \\ \mathsf{dov} & \text{otherwise} \end{cases}$$
$$dp(\mathsf{ooa}(\Delta_1, \ldots, \Delta_m)) = \begin{cases} dp(\Delta_i) & \text{if } (\Delta\text{:P and } \Delta_i\text{:P) or } (\Delta\text{:D and } \Delta_i\text{:D}) \\ \mathsf{ooa} & \text{otherwise} \end{cases}$$
$$dp(\mathsf{sm}(\Delta_1, \ldots, \Delta_m)) = \mathsf{sm}$$

Table 2: Decision point for sample combining algorithms

Anything below the decision point should be included in the formulation of the explanation as 'real' decisions are being made. It is worth noting that users should be able to understand their positions relative to the decision point. In particular, we assume a user knows how its policy fits in the policy hierarchy. Thus, any ancestor node of a user policy should be recognizable by the user. To ensure the explanation is formulated from a point that is recognizable by the user, we start from one such ancestor node. In particular, we start from the least node (as we would like explanations to only give relevant information) that satisfies both properties above, i.e. that is an ancestor of the user policy and decision point. We call this node the *evaluation point*.

Definition 12. *The* evaluation point *(for a user u) is the least node that is an upper bound of both the visible decision point and of policy node $n[u]$.*

Note that the least element is well defined as there always exists one (the root) and the set of upper bounds of a node (the decision point) is totally ordered.

We formulate the feedback starting from the evaluation point. We define functions msg_N, which gives the description of a given node, and msg_T, which gives the description of a subtree starting from its root node. We assume that each basic element e has a string representation, which we denote by $e.name$. For a node, the description expresses the decision reached:

$$msg_N(n[e]\text{:}d) = \begin{cases} e.name \text{ denied} & \text{if } d = \mathsf{Deny} \\ e.name \text{ permitted} & \text{if } d = \mathsf{Permit} \\ e.name \text{ failed to reach a decision} & \text{if } d = \mathsf{Indeterminate} \\ e.name \text{ did not apply} & \text{if } d = \mathsf{NA} \end{cases}$$

This generic text can be customized by considering the type of element and combining algorithms involved. For instance, we can state: "$e.name$ voted to deny" for a node $\mathsf{sm}[\mathsf{e}]$:D rather than the more generic "$e.name$ denied". (For reasons of space, we do not list all customizations considered.)

For a tree we start with the description of the root node and recursively add the explanation of its subtrees. Note that visibility constraints could theoretically give a situation in which some but not all of the children are visible to the user and the visible children do not constitute an explanation (according to the combining algorithm) of the decision reached by the node. As showing this 'incomplete explanation' would be confusing to the user, they are not included in the textual explanation in this case. Specifically:

Algorithm 1: $msg_U(n_d[x], n_e)$

if $n_d[x]$ *is (within) an archetype of the user* **then**
  | "Your archetype" + $msg_T(n_e)$
else if $n_d[x]$ *is (within) a level of the user* **then**
  | "Your level" + $msg_T(n_e)$
else if $n_d[x]$ *is (within) a level higher than any level containing the user* **then**
  | "Your decision was overruled by $x.name$:" + $msg_T(n_e)$
else if $n_d[x]$ *is (within) a level lower than some level containing the user* **then**
  | "You failed to overrule the decision of $x.name$:" + $msg_T(n_e)$.
else
  | "The decision of " + $x.name$ + "was followed:" + $msg_T(n_e)$.

$$msg_T(n) = \begin{cases} msg_N(n) + \text{“because”} \\ \qquad +msg_T(n_1) + \ldots + msg_T(n_m) & \text{if } n_1, \ldots, n_m \text{ visible children of } n \text{ explaining the decision} \\ msg_N(n) + \text{“.”} & \text{otherwise} \end{cases}$$

Also here we can make customizations to further improve the readability of the explanation. For example, for archetype node $n[a]{:}d$ with children $n_1{:}\, d_1, \ldots, n_m{:}\, d_m$, we can use short hand "(" + $u_1.name : d_1$ + "," + ... + $u_m.name : d_m$ + ")" which simply lists the decisions of the user's involved rather than using the generic 'because' format resulting in much more compact explanations.

With the description of a subtree in place, we can now define the textual description given to the user, which captures the visible decision point ($n_d[x]$) and evaluation point (n_e), and a description of the subtree starting at n_e:

$$msg_U(n_d[x], n_e) = \text{“The decision of ” + } x.name \text{ + ”was followed: ” + } msg_T(n_e)$$

As before we consider customizations that enhance specific (common) cases as shown in Algorithm 1.

Example 6. Consider the justification computed in Example 5. The textual explanation for the mismatch of user C is:

Your archetype DS voted to deny (A:Deny, B:Deny, D:Deny).

For user G the textual explanation is:

Your decision was overruled by DS: sub-hierarchy at level ℓ_1 denied because DS voted to deny.

User G's explanation contains both the decision of DS and an explanation of how this decision overwrites his choice; this happens at the point where 'sub-hierarchy at level ℓ_1' denies.

Finally, social network SN does not get an explanation, only the decision.

5 Related Work

Recent years have seen an increasing interest in access control for collaborative systems and, in particular, for social networks. This interest has resulted in several access control solutions (e.g., [3, 9, 26]) that aim to regulate the exchange of information between collaborative users. These solutions are complementary to our work as they consider different aspects of collaborations. Within these solutions access decisions are usually made based on the interpersonal relationships between the resource owner and the resource requester, while assuming that resources are owned by a single entity. Moreover, these solutions only focus on the specification and enforcement of access control policies for collaborative systems and do not address the problem of transparency of access decision making.

Some social networks provides basic functionality for transparency. For instance, Linkedin allows its users to view their profile from the perspective of their connections and their public profile. Similarly, FaceBook provides a "View As" functionality that allows users to visualize their profile from another user's perspective. This functionality, however, can provide users with a misleading feeling of control over their information [7].

The detection of policy conflicts is largely addressed in the area of formal policy analysis. For instance, change-impact analysis [8] aims to extract the differences between two policies. Backes et al. [2] propose a notion of policy refinement in which a policy refines another policy if, whenever the latter returns Permit (or Deny), the first policy returns the same decision. Hughes and Bultan [14] present a stronger notion of policy refinement called policy subsumption. In addition to impose constrains on Permit and Deny decisions as in policy refinement, subsumption also imposes constraints on the Indeterminate decision. Turkmen et al. [28] propose a formal framework for policy analysis based on SMT. This framework allows the verification of XACML policies against a number of well-known security properties including change-impact, policy refinement and subsumption. These frameworks, however, aims to support policy authors in the definition of their policies and are not suitable for run-time analysis. Moreover, they only indicate if two policies are conflicting (possibly along with a counterexample indicating the conflict), but do not provide a justification for the conflict.

A few proposals address the problem of transparency in collaborative systems by providing feedback about policy conflicts to the entities governing the data. For instance, Hu et al. [13] present an authorization analysis tool for examining over-sharing and under-sharing of shared resources in social networks. Mahahmudlu et al. [20] proposes a notification mechanism that determines at run time the type of conflicts that occurred (e.g., DenyButPermit, PermitButDeny). In particular, users can declare the type(s) of conflicts they are interested in and only be notified about those conflicts. Although these solutions make a first step towards user awareness, the feedback provided to users only indicates if their policies have been overridden. This work makes a step further by providing users with an explanation of why their policies have been overruled.

KNOW [17] and Cue [10] provide feedback suggesting a requester possible alternatives to access the data (e.g., changing role). Similarly to our work, feedback is protected through the use of meta policies, thus ensuring that a desired level of confidentiality is preserved. However, the goal of these frameworks is orthogonal to our work. While KNOW and Cue aim to inform users why their access requests have not been granted, we aim to explain users why their policies have been overridden.

6 Conclusion

This paper presented an approach to enhance user understanding in the access decision making process when policy mismatches occur. In particular, we proposed an approach to compute justifications explaining users why their decisions were overruled. To determine at which granularity a user can see the justification, we use visibility restrictions that indicate the portion of the justification visible to the user based on its place in the archetype hierarchy and visibility policies. We also showed how the feedback can be formulated in a human readable format, focusing on the relevant parts and customizing the feedback to reflect the relations of the user with the data.

As future work, we plan to integrate the feedback mechanism proposed in this work into existing XACML-based authorization solutions. This requires extracting the decision annotated evaluation path from an XACML response. We envision this can be done by exploiting the `<ReturnPolicyIdList>` element, which is used to request an XACML policy decision point to return the list of applicable policies and policysets that were used to obtain the decision [23]. Users policies can be sensitive and, thus, not all users may be allowed to see the full explanation of the decision. In this work, we addressed this problem by restricting the visibility of the feedback disclosed to users, depending on their relation with the data and visibility policies. However, also access requests, which eventually have to be disclosed along with the feedback, might provide insights into the policies of other users. Moreover, one might consider requests themselves to be 'sensitive' (for privacy) irrespective of what they reveal about the policies. Thus, it is desirable to give only a minimal amount of attributes that reveals the mismatch rather than the actual request. To this end, we plan to extend visibility restrictions to access requests, thus preventing a user to learn information that policy authors may wish to not reveal as well as to protect requester's privacy as much as possible. In this work we demonstrated the feedback mechanism through a typical scenario in FaceBook-like social networks. User studies to evaluate its impact on user awareness is left as future work.

Acknowledgments: This work has been partially funded by the ITEA2 projects FedSS (No. 11009) and M2MGrid (No. 13011), the EDA project IN4STARS2.0, and the Dutch national program COMMIT under the THeCS project.

References

1. Ashley, P., Hada, S., Karjoth, G., Powers, C., Schunter, M.: Enterprise Privacy Authorization Language (EPAL 1.2) (2003)
2. Backes, M., Karjoth, G., Bagga, W., Schunter, M.: Efficient comparison of enterprise privacy policies. In: Proc. of Symposium on Applied Computing. pp. 375–382. ACM (2004)
3. Carminati, B., Ferrari, E.: Collaborative access control in on-line social networks. In: Proceedings of International Conference on Collaborative Computing. pp. 231–240 (2011)
4. Costante, E., den Hartog, J.I., Petkovic, M.: On-line trust perception: What really matters. In: Proc. of Workshop on Socio-Technical Aspects in Security and Trust. pp. 52–59. IEEE (2011)
5. Costante, E., den Hartog, J.I., Petkovic, M.: Understanding perceived trust to reduce regret. Computational Intelligence 31(2), 327–347 (2015)
6. Damen, S., den Hartog, J., Zannone, N.: CollAC: Collaborative access control. In: Proceedings of International Conference on Collaboration Technologies and Systems. pp. 142–149. IEEE (2014)

7. Damen, S., Zannone, N.: Privacy implications of privacy settings and tagging in facebook. In: Secure Data Management. pp. 121–138. LNCS 8425, Springer (2013)
8. Fisler, K., Krishnamurthi, S., Meyerovich, L.A., Tschantz, M.C.: Verification and Change-impact Analysis of Access-control Policies. In: Proceedings of International Conference on Software Engineering. pp. 196–205. ACM (2005)
9. Fong, P.W.: Relationship-based Access Control: Protection Model and Policy Language. In: Proc. of Conf. on Data and Application Security and Privacy. pp. 191–202. ACM (2011)
10. Ghai, S.K., Nigam, P., Kumaraguru, P.: Cue: A Framework for Generating Meaningful Feedback in XACML. In: Proceedings of Workshop on Assurable and Usable Security Configuration. pp. 9–16. ACM (2010)
11. Guarda, P., Zannone, N.: Towards the development of privacy-aware systems. Information & Software Technology 51(2), 337–350 (2009)
12. den Hartog, J., Zannone, N.: A policy framework for data fusion and derived data control. In: Proceedings of the ACM International Workshop on Attribute Based Access Control. pp. 47–57. ACM (2016)
13. Hu, H., Ahn, G.J., Jorgensen, J.: Multiparty Access Control for Online Social Networks: Model and Mechanisms. TKDE 25(7), 1614–1627 (2013)
14. Hughes, G., Bultan, T.: Automated verification of access control policies using a SAT solver. Int. Journal on Software Tools for Technology Transfer 10(6), 503–520 (2008)
15. Jajodia, S., Samarati, P., Sapino, M.L., Subrahmanian, V.S.: Flexible Support for Multiple Access Control Policies. ACM Trans. Database Syst. 26(2), 214–260 (2001)
16. Kaluvuri, S.P., Egner, A.I., den Hartog, J., Zannone, N.: SAFAX – An Extensible Authorization Service for Cloud Environments. Frontiers in ICT 2(9) (2015)
17. Kapadia, A., Sampemane, G., Campbell, R.H.: KNOW Why Your Access Was Denied: Regulating Feedback for Usable Security. In: Proceedings of Conference on Computer and Communications Security. pp. 52–61. ACM (2004)
18. Lamport, L.: How to write a long formula. Formal Aspects of Computing 6(5), 580–584 (1994)
19. Li, N., Wang, Q., Qardaji, W., Bertino, E., Rao, P., Lobo, J., Lin, D.: Access control policy combining: Theory meets practice. In: Proc. of SACMAT. pp. 135–144. ACM (2009)
20. Mahmudlu, R., den Hartog, J., Zannone, N.: Data governance & transparency for collaborative systems. In: Data and Applications Security and Privacy. LNCS, Springer (2016)
21. Matteucci, I., Mori, P., Petrocchi, M.: Prioritized execution of privacy policies. In: DPM/SETOP. pp. 133–145. LNCS 7731, Springer (2012)
22. OASIS XACML Technical Committee: eXtensible Access Control Markup Language (XACML) Version 2.0 (2005)
23. OASIS XACML Technical Committee: eXtensible Access Control Markup Language (XACML) Version 3.0 (2013)
24. Paci, F., Zannone, N.: Preventing information inference in access control. In: Proceedings of Symposium on Access Control Models and Technologies. pp. 87–97. ACM (2015)
25. Reeder, R.W., Bauer, L., Cranor, L.F., Reiter, M.K., Vaniea, K.: Effects of access-control policy conflict-resolution methods on policy-authoring usability. CyLab p. 12 (2009)
26. Squicciarini, A.C., Paci, F., Sundareswaran, S.: PriMa: a comprehensive approach to privacy protection in social network sites. Annales des Télécommunications 69(1-2), 21–36 (2014)
27. Trivellato, D., Zannone, N., Etalle, S.: GEM: A distributed goal evaluation algorithm for trust management. TPLP 14(3), 293–337 (2014)
28. Turkmen, F., den Hartog, J., Ranise, S., Zannone, N.: Analysis of XACML policies with SMT. In: Principles of Security and Trust. pp. 115–134. LNCS 9036, Springer (2015)
29. Winsborough, W.H., Seamons, K.E., Jones, V.E.: Automated trust negotiation. In: Proceedings of DARPA Information Survivability Conference. pp. 88–102 (2000)