HAL
archives-ouvertes.fr

Design Patterns for Multiple Stakeholders in Social Computing

Pooya Mehregan, Philip Fong

▶ To cite this version:

Pooya Mehregan, Philip Fong. Design Patterns for Multiple Stakeholders in Social Computing. 28th IFIP Annual Conference on Data and Applications Security and Privacy (DBSec), Jul 2014, Vienna, Austria. pp.163-178, 10.1007/978-3-662-43936-4_11 . hal-01284852

HAL Id: hal-01284852
https://hal.inria.fr/hal-01284852

Submitted on 8 Mar 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution| 4.0 International License

Design Patterns for Multiple Stakeholders in Social Computing

Pooya Mehregan and Philip W. L. Fong

Department of Computer Science
University of Calgary
Calgary, AB, Canada
{pmehrega, pwlfong}@ucalgary.ca

Abstract. In social computing, multiple users may have privacy stakes in a content (e.g., a tagged photo). They may all want to have a say on the choice of access control policy for protecting that content. The study of protection schemes for multiple stakeholders in social computing has captured the imagination of researchers, and general-purpose schemes for reconciling the differences of privacy stakeholders have been proposed. A challenge of existing multiple-stakeholder schemes is that they can be very complex. In this work, we consider the possibility of simplification in special cases. If we focus on specific instances of multiple stakeholders, are there simpler design of access control schemes? We identify two design patterns for handling a significant family of multiple-stakeholder scenarios. We discuss efficient implementation techniques that solely rely on standard SQL technology. We also identify scenarios in which general-purpose multiple-stakeholder schemes are necessary. We believe that future work on multiple stakeholders should focus on these scenarios.

Keywords

Social Computing, Privacy, Multiple Stakeholders, Discretionary Access Control, Owner, Controller, Design Pattern.

1 Introduction

The advent of social computing has brought about fundamental changes in our understanding of Discretionary Access Control (DAC). In traditional DAC, such as the Graham-Denning model [6, 12], every object is associated with a distinguished user known as the ***owner*** of that object. Ownership in DAC is not about property rights. Rather, the owner is the user who has full administrative privileges over that object: i.e., that user is granted the privileges to administrate the access control policies of the resource. Every object has exactly one owner, though ownership is transferrable. The owner may selectively delegate some administrative privileges to other users known as the ***controllers*** of the object. This classical picture requires revision in the face of the new privacy needs of social computing.

In social computing, users often annotate contents that are originally contributed by others (e.g., commenting, liking). At other times, contents come to be associated with users other than the original contributors (e.g., photo tagging). There are also scenarios in which multiple users co-contribute a piece of information (e.g., friendship articulation). In all these scenarios, multiple users have a privacy stake in a given content: they all have an interest in determining the access control policy for the content. The classical picture of one owner delegating administrative duties to trusted controllers is no longer valid. Different stakeholders of the same content now have diverse privacy preferences, and they do not necessarily agree with one another. Yet, existing social computing systems still insist that the visibility of a content is controlled by a unique "owner". The privacy shortcomings of such a practice have been well-documented [19].

Squiccinarini *et al.* [15, 16] were the first to identify the need to take into account the often diverse privacy preferences of all stakeholders when the access control policy of a resource is to be selected. Subsequent works (e.g., [19, 21]), especially the seminal contributions of Squiccinarini *et al.* [15–17] and Hu *et al.* [7, 9, 8], have firmly established the necessity and feasibility of access control schemes that reconcile the diverse protection needs of multiple stakeholders. Such schemes have come to be called by different names, such as co-ownership [17, 16, 15], collaborative privacy management/control [17, 15, 8], and multiparty access control [9, 7]. For the sake of neutrality, we choose to call this phenomenon ***multiple stakeholders***. Many of the proposed schemes for multiple stakeholders are very general, equipped with conflict resolution mechanisms that are not easy to understand by a regular user.

In this paper, we raise the question of *simplification*: Are there cases in which general-purpose multiple-stakeholder schemes are *overkill*? If we focus on specific instances of multiple stakeholders (e.g., liking, photo tagging, etc), can we honor the diverse privacy preferences of the stakeholders through *simple* designs of access control schemes? Our answers to these questions are univocally affirmative. Our goal is not to question the value of general-purpose schemes for multiple stakeholders. (There are instances in which general-purpose schemes are absolutely irreplaceable, as we shall see in §8.) Rather, our goal is to sharpen future discussions of multiple stakeholders, and to put forward access control schemes that can be used today, by social computing vendors such as Facebook and Google+. Specifically, our contributions are the following:

1. We propose two design patterns [5] for addressing a large number of multiple-stakeholder instances (§3, §4). Out of the five examples of multiple stakeholders that are quoted in the literature, our design patterns can address the privacy needs of three of them.
2. We identify previously unpublished privacy breaches in some of the above instances of multiple stakeholders, and propose ways to prevent them (§5).
3. We propose an implementation strategy for the design patterns that rely solely on standard SQL technology (§6), and demonstrate that the resulting performance meets the responsiveness requirements of web applications (§7).

4. We carefully identify scenarios in which general-purpose schemes of multiple stakeholders are needed (§8).

2 Related Work

The first work that identifies and addresses the problem of multiple stakeholders is that of Squicciarini *et al.* [15, 16]). In their proposal, whenever the privacy policy of a co-administrated object is to be decided, stakeholders are asked to take part in an auction. Stakeholders earn credits by creating objects and sharing their ownerships with other users. Using their credits, the stakeholders give their highest bid for the privacy policy they want to be selected for the co-administrated object. Then, the highest bid wins the auction and the privacy policy associated with the winning bid is adopted. The stakeholders then get taxed from their credits, with amounts depending on how dominant their roles were in determining the outcome of auction (winning privacy policy). The auction is based on a mechanism called Clarke-Tax, which in turn is a special case of Vickrey-Clarke-Groves (VCG). Inference techniques based on Folksonomies have also been proposed to prevent repetition of auctions for similar objects which are co-administrated by the same stakeholders. The biggest drawback for this work is its low usability. Users will have difficulty figuring out how the auction works and how to translate their privacy to a bid amount. The need for usability is later addressed in a subsequent work of Squicciarini *et al.* [17], in which majority-voting is used in place of complex auctions.

The work in [19] not only points out the problem of multiple stakeholders (that some stakeholders do not have control over the objects for which they have a privacy stake), but also demonstrates concretely its consequences. Specifically, the authors demonstrated how inference attacks may result from not addressing the privacy preferences of stakeholders. They also considered a simple solution in which the conjunction of the stakeholders' privacy preferences are taken as the access control policy for the co-administrated object.

In [21], stakeholders collaborate in authoring a policy for a co-administrated object. The policy can be edited by each of the stakeholders, with the following restrictions. Every policy is divided into two parts: (a) the ***weak conditions*** and (b) the ***strong conditions***. Weak conditions are *negotiable* conditions of access. Each stakeholder can freely modify the weak conditions, even if they are contributed by other stakeholders. The strong conditions are *non-negotiable* conditions of access. When a stakeholder contributes a strong condition, the authorship is recorded. Only the author of a strong condition can revise it. Conflict resolution is performed manually. For example, when the strong conditions become overly restrictive for a stakeholder, there is the option for revising the object itself (e.g., blurring parts of a picture) in order to inspire other stakeholders to relax their strong conditions.

In the seminal work of Hu and Ahn [7, 9], a comprehensive requirement analysis for the problem of multiple stakeholders is given. Different types of controllers (i.e., stakeholders) are distinguished: the ***owner*** is the user whose profile is host-

ing the co-administrated object; the ***contributor*** is the user who contributes the co-owned object to the owner's profile; ***stakeholders*** are users who have been "tagged" in the co-administrated object; ***distributers*** are users who have re-shared the co-administrated object on their profiles. Each stakeholder specifies for the co-administrated object her preferred privacy policies as well as a sensitivity level. The latter is a normalized quantity representing the perceived privacy sensitivity of the object. Obviously, the privacy policies specified by the various controllers may not agree with one another. This is the first work that draws connection between conflict resolution and policy composition. Voting schemes have been proposed for resolving conflicts among the policies specified by the controllers.

In [8], another scheme for conflict resolution has been proposed. The fundamental assumption is that there is a trade-off between the need for privacy and the desire to share. The two are operationalized into two corresponding quantities: privacy risk and sharing loss. A quantitative scheme is devised to trade off the two quantities.

In the works above, the various instances of the multiple-stakeholder problem are treated in a uniform manner, hence a generic solution is proposed for all the problem instances. This approach does not take into account the idiosyncrasies of different instances of multiple stakeholders, which occasionally admit straightforward and efficient solutions. This is the topic to which we now turn.

3 Design Pattern: Simple Annotations

A ***design pattern*** is a reusable software design for a recurring software design problem [5]. In this and the next section, we discuss respectively two design patterns for two well-defined families of multiple-stakeholder scenarios. This section presents a design pattern known as ***Simple Annotations***.

3.1 Setting

Social computing systems support not only the sharing of contents, but also further social interactions that are prompted by the initial sharing of contents. Examples of such social interactions include commenting, liking, tagging and resharing. We use the term ***annotations*** to refer to these secondary contents that are associated with a shared content. Annotations that are not further annotated are said to be ***simple***. The more complex subject of ***higher order annotations*** (i.e., annotations of annotations) is discussed in the next section.

The author of an annotation can be different from the author of the annotated content. When an annotated content is displayed, the annotations are displayed along with it. In mainstream social computing systems, the author of the annotated content is taken to be the DAC owner of the ***content-annotations aggregate***: i.e., the author of the annotated content is the one who can specify the privacy setting of both the original content and its annotations.

3.2 Privacy Challenges

The problem of the mainstream design is that *the authors of annotations also have privacy stakes in the visibility of the content-annotations aggregate, and yet they have absolutely no say in the visibility of the annotations that they authored.* In fact, we accept the following as a general design principle in the context of multiple stakeholders.

Design Principle 1 *Every stakeholder of a content shall have a say on the access control policy that protects the content.*

The users of Facebook came to notice this issue when the Ticker [18, 10, 20] (a real-time news feed which appears at side of the Facebook page) started showing the friends of users their activities such as what they have liked, commented and shared. Facebook claims that, Ticker is not breaching users' privacy since no privacy setting has been changed and the information showed in Ticker is already there and visible by those who can view it in the Ticker. That is, the privacy settings of the annotated (i.e., liked, commented, or shared) contents already allow access to the the content-annotations aggregate. Below is Facebook's announcement regarding the privacy of News Feed and Ticker [4]:

> *"People included in the audience of the post can see your comment or like in News Feed or ticker as well as other places around Facebook. You can check who something is shared with by going to the post and hovering over the audience icon."*

What Facebook fails to appreciate is that the privacy preferences of annotation authors are not honoured. When a user likes a content, she has absolutely no control over who may or may not be able to see that she likes the content. In the following, we explain in concrete terms how the privacy of annotation authors are breached in the cases of liking, tagging and (re)sharing.

Liking. In Facebook, users can *"like"* (also *"+1"* in Google+) a content to show their support for, affirmation of, or interest in that content. When the content is displayed, a total number of "likes" is also displayed, and the viewer of the content may also follow a link to display the full list of users who have liked the content. When the list is displayed, the following information of each liker is displayed: (a) display name, (b) thumbnail picture, and (c) link to profile.

A user who expresses her affirmation of a content may want the "like" to be displayed with discretion. For example, liking a political commentary may lead to troubles in certain countries, and yet, expressing such affirmations is an important democratic expression. Currently, there is no mechanism in Facebook that would allow the liker to control the visibility of his or her likes.

Tagging. Users can *tag* one another in today's social network systems. Facebook, Google+ and Instagram all have this feature. Specifically, users can tag one another in contents such as pictures, videos and textual information like statuses,

comments and captions, usually by means of a mention tag *'@'* followed by the display name of a user.

The tagging of photos has been a classical example for multiple stakeholders in the literature [7, 9, 8, 15, 17, 19]. When a user uploads a photo in which other users appear, the former is disclosing potentially sensitive information about the latter. This by itself is a privacy issue, but this is not an instance of the multiple-stakeholder problem. It becomes a multiple-stakeholder scenario when the latter users are tagged by the former user — when the identities of these users are explicitly associated with the photo. These tags will be displayed together with the photo. It was not long ago that Facebook introduced a privacy setting that requires users to ask for other users' consent before they can get them tagged in a content.

Sharing. Facebook users can reshare a content that is originally posted by another user. There are two types of reshares in Facebook: (a) ***link reshares*** and (b) ***content reshares***.

In link reshares, a user u posts a URL l along with a caption. A viewer v of the posting can reshare l (without the caption). There are two ways in which user privacy can be breached. First, the original posting of l by u shows both the total number of reshares, as well as a link that lists all the resharers. When this link is followed, the identity of v will be disclosed. The situation is analogous to that of liking. Second, the resharing of l by v is displayed with the phrase "via u", which discloses the identity of u. The situation is analogous to tagging.

In content reshares, a user u uploads some content c (e.g., photo) to Facebook. A viewer v of c can reshare c. Again, there are two privacy concerns in play here. First, the identity of v appears in a list associated with the posting of c by u. As noted above, this is analogous to liking. Second, u is clearly a stakeholder for the resharing of c by v. User u specifies a policy $p_{u,c}$ for controlling access to c, and v specifies a policy $p_{v,c}$ for controlling access to the reshared c. A user w can access the reshared c when $p_{u,c} \wedge p_{v,c}$ is satisfied.

3.3 Solution: Separation of Protection

Previous works in multiple stakeholders take the content-annotations aggregate as an indivisible entity, and thus attempt to address the multiple-stakeholder problem at that level. Our use of the term "aggregate" to refer to a content and its annotations is intended to make explicit the fact that we are not dealing with an atomic entity, but rather a composite one. Recognizing this, we articulate the following design principle.

Design Principle 2 *If every component of an aggregate entity can have a different set of stakeholders, then each component should be protected separately by a different access control policy.*

The applicability of the above principle depends on the allocation of stakeholders.

Notice that a stakeholder of an annotated content also has a privacy stake in its annotations, for the latter convey information about the former.

Design Principle 3 *Every stakeholder of an annotated content is a stakeholder of its annotations.*

In short, an annotation ***inherits*** all the stakeholders of the content to which it annotates. These stakeholders are called ***inherited stakeholders***. Stakeholders of a content that are not inherited are called ***principal stakeholders***.

Consider, for instance, a hypothetical social computing system in which every content has exactly one principal stakeholder, namely, the author of that content. Applying Design Principle 3, a simply annotated content has exactly one stakeholder (i.e., the content's author), while a simple annotation has two stakeholders (i.e., the annotated content's author and the annotation's author). Since every component of a content-annotations aggregate has a different set of stakeholders, Design Principle 2 mandates that each must be protected by a separate policy.

How then are we to assign an access control policy to each component? Our goal is to simplify the design of protection schemes for multiple stakeholders. Consequently, we make two design choices that are aimed at producing simple and yet effective protection. The first decision is to minimize the effort of policy specification that needs to be performed by a user.

Design Decision 4 *Every stakeholder u of content c has a* **preferred policy** *$p_{u,c}$ that expresses the privacy preference of u for c. If u is a principal stakeholder of c, then u will explicitly specify $p_{u,c}$. Otherwise, u is an inherited stakeholder of c, c is an annotation of some content c', and u is also a stakeholder of c': then $p_{u,c} = p_{u,c'}$.*

In short, preferred policies are ***inherited*** by annotations. A second design choice is to realize Design Principle 1 by simple conjunction of preferred policies.

Design Decision 5 *Suppose $S_c = \{u_1, \ldots, u_k\}$ is the set of stakeholders for content c, and their preferred policies for c are $p_{u_1,c}$, $\ldots$, $p_{u_k,c}$. Then the access control policy p_c for content r is $\bigwedge_{u \in S_c} p_{u,c}$.*

We use the following examples to illustrate how the design works out in practice.

Liking. The author of a content c that can be liked will specify a preferred policy p_1 for c. Because the content author is the sole and principal stakeholder of the content, the visibility of c is controlled solely by p_1. When another user "likes" c, she will be given the opportunity to specify a preferred policy p_2 for this like entry. The access control policy for this like entry is $p_1 \wedge p_2$.

When c is displayed, the total number of likes will be displayed, together with a link for displaying the "likers". When that link is followed, *not all likers are displayed*. The system will check the access control policy of each like entry, and display only those that are accessible by the viewer.

Tagging. In the same vein, every tag is protected separately from the content to which the tag belong. The content itself is protected solely by the preferred policy of its author, who is the principal stakeholder of the content. The principal

stakeholder of a tag is the user identified by the tag. This user will specify a preferred policy for the tag. The author of the original content is an inherited stakeholder of the tag. Consequently, each tag is protected by the conjunction of two preferred policies: (a) the preferred policy of the content, and (b) the preferred policy of the user who is being tagged. When the content is displayed, only a subset of its tags are displayed. Tags for which the access control policy is not satisfied are not displayed. To simplify policy specification, a user may have a default policy for controlling the visibility of tags that identify her.

Sharing. Recall the three kinds of privacy concerns surrounding link and content resharing. First, the listing of resharers along with the original posting is analogous to liking, and thus can be handled by a scheme like the one we proposed above for liking. Second, the "via" clause in a reshared link behaves like a tag, and thus it can be handled by a scheme like the one we proposed above for tagging. Third, a reshared content is protected in Facebook using exactly the same design as outlined in Design Decisions 4 and 5, which speak to the robustness of these two design decisions.

4 Design Pattern: Higher Order Annotations

The design pattern we present in this section handles ***higher order annotations***. That is, an annotation can be further annotated. The classical example of such higher order annotations is commenting. A posting in a forum can be annotated by comments, which in turn can be further commented on.

4.1 Replying Comments

In most of today's social network systems and online communities, users are able to leave comments on the contents created by themselves or other users. One type of comments mimic the structure of emails. In this type of comments, a user explicitly selects the content that her comment replies to (just like replying to an email). This is a common practice in forums and online communities. Therefore, comments of this type constitute a tree-like structure with the original content as the root of the tree and different threads of comments become branches of the tree. We call this type of comments ***replying comments*** because of their resemblance to replying emails. Facebook has added replying comments in the posts that have several hundreds of comments. However, the depth of the tree of comments cannot grow more than two in this case.

Design Principles 1, 2 and 3, and Design Decisions 4 and 5 all apply to this setting. Suppose c_0, c_1, ..., c_k is a thread of contents, such that c_0 is a non-annotation content (root), and c_i is annotated by c_{i+1}. Let u_i be the author (and thus principal stakeholder) of c_i. Then the stakeholders of c_k are u_0, u_1, ..., u_k. Each u_i must explicitly specify a preferred policy p_i for c_i. Preferred policies are inherited, and thus the access control policy for c_k is the conjunction $\bigwedge_{i=0}^{k} p_i$.

4.2 Appending Comments

Replying comments do not cover all types of commenting mechanisms in social computing. A notable exception is the mechanism that we call ***appending comments***, which is widely deployed in many social network systems. A comment that a user creates gets appended to the end of all the existing comments for the original contents (hence appending comments). Unlike replying comments, this type of comments has less structure than replying comments, and thus it makes the allocation of stakeholders more ambiguous. In the worst case, a newly introduced appending comment may be (implicitly) responding to all the existing comments (and thus annotating all preceding comments as well as the original content). Consequently, rather than a tree structure, the original content and its appending comments form a sequence $c_0, c_1, \ldots, c_k$, where c_0 is the original content, and $c_1, \ldots, c_k$ are the appending comments.

Applying Design Principles 1, 2 and 3, and Design Decisions 4 and 5 to this situation yields the following. Suppose u_i is the author (and thus principal stakeholder) of c_i, and p_i is the preferred policy explicitly specified by u_i for c_i. The stakeholders of c_i are $u_0, u_1, \ldots, u_i$. Therefore, the access control policy for c_k will be the conjunction $\bigwedge_{i=0}^{k} p_i$. Note the difference between this conjunction and the one for replying comments. In the case of replying comments, the access control policy of a comment is the conjunction of the preferred policies of the *ancestors* of that comment. In the case of appending comments, the access control policy of a comment is the conjunction of the preferred policies of all the *preceding* comments.

4.3 Hybrid Solution for Comments

The scheme proposed above for appending comments has a drawback analogous to a well-known problem in Low Watermark Model of Biba [2]. The accessibility of comments becomes increasingly restrictive as users create more and more comments: if a user is able to view a highly restrictive comment (restrictive in terms of access control policy), then this user will not be able to leave a comment with a less restrictive access control policy.

To overcome this drawback, we propose a hybrid solution, in which a user may annotate a content by either appending comment or replying comment. Comments are by default appending comments. The author of an appending comment implicitly consents to adopting the most liberal preferred policy (i.e., everyone). Consequently, the access control policy of an appending comment will be the same as the access control policy of the content to which the appending comment is annotating. If a user wants to explicitly specify a preferred policy, then the user may introduce a replying comment (she will need to point to a specific comment to which she is replying). This preferred policy will not affect the accessibility of the appending comments at a higher level. This prevents the low-watermark effect of pure appending comments, but also provides flexibility of protection offered by replying comments. It is easy to add this feature to an existing social computing system that features appending comments.

5 Relationship Disclosure via Annotations

Annotations create a channel by which user relationships can be inferred. Facebook (also Google+) discloses the "audience" of a content to its viewers. The "audience" is essentially the access control policy of the content. Suppose users u and v prefer to hide their friendship from other users. To that end, they set the accessibility of their friend lists to "*only me*". Suppose further that u shares a content c with *friends*, and subsequently v likes c, but v sets the preferred policy of the like to "everyone". Suppose now an observer w comes along. User w is a friend of u, and thus w can view c. When w examines the audience of c, w becomes aware that only friends of u can view c. User w then notices that v likes c. Now w can infer that v is a friend of u. What u and v are not aware is that simply by making c and its annotations visible could lead to the disclosure of their relationships.

The above inference is possible because w can identify the audience of c. We believe that Facebook (also Google+) discloses the audience of a content in order to warn annotators of the visibility of the content. Our solution of protecting an annotated content and its annotations separately (§3 and §4) removes the need for disclosing the "audience" of a content. Without knowing the exact access control policy, the attacker cannot infer relationships with certainty.

Suppose we are paranoid, and we worry that the observer w may be able to guess that the access control policy of the above content is "*friends*" (maybe by observing that other contents of u are usually protected by the "*friends*" policy). Then relationship disclosure will still be possible. We propose here another solution which tackles this paranoia. Suppose p_c is the access control policy of a content c that is created by user u. Suppose $p_{v,a}$ is the preferred policy of an annotation a of c, where a is contributed by v. Suppose p_f is the access control policy of the friend list of u. Then we set the access control policy p_a of annotation a to be $p_c \wedge p_{v,a} \wedge p_f$. In general, relationship inference can be prevented if the access control policy that protects an annotation (p_a) is at least as restrictive as the one protecting the relationship (p_f).

6 Implementation Strategy

The two design patterns presented in §3 and §4 refrain from displaying all annotations (as is done in existing social computing systems). Instead, each annotation is guarded by a separate access control policy, and only the accessible annotations are displayed. This last feature calls for special implementation techniques.

Open Accessibility Queries. A typical Policy Decision Point (PDP) must perform what we call ***definite accessibility checks*** in order to test whether a given requestor may access a *given* resource. To list the annotations that are accessible by a requestor, a naive implementation will make a database query to collect all annotations, and then procedurally iterate through the annotations, filtering away the ones that fail the accessibility check. Such an implementation is likely unacceptable in performance.

Table	Columns	Indexes
Friends	ID (int)	Clustered: ID (Primary Key)
	UserID1 (int)	Non-Clustered: UserID1 and UserID2
	UserID2 (int)	Non-Clustered: UserID1 Include UserID2
		Non-Clustered: UserID2 Include UserID1
Resources	ResourceID (int)	Clustered: ResourceID (Primary Key)
	PolicyID (int)	Non-Clustered: ParentID and RootID Include ResourceID
	OwnerID (int)	Non-Clustered: OwnerID and ParentID
	ParentID (int)	Non-Clustered: ParentID
	RootID (int)	Non-Clustered: PolicyID and ParentID Include ReaourceID and OwnerID
		Non-Clustered: RootID Include ResourceID
Users	UserID (int)	Clustered: UserID (Primary Key)

Fig. 1. Database tables and their columns and indexes

A more efficient implementation will push the work of accessibility filtering to the database management system, which is equipped with highly efficient indexing and query optimization technologies. In essence, we need to be able to evaluate ***open accessibility queries***: *Given a requestor, find all the accessible resources of a certain kind (e.g., annotations of a given content).*

There are two variations to this query: one involving only simple annotations, and the other involving higher order annotations. We present in the following the high-level idea of how open accessibility queries can be answered in each case, using solely standard SQL technologies.

Modelling a Social Network System. Figure 1 shows the relational database tables that we use as basis for articulating our implementation strategy. Real implementations will probably contain more details, but we believe our tables capture the essence of a social network system. The *Users* table tracks user identifiers. The *Friends* table captures friendship among users. The *Resources* table captures resources, their preferred policies (enumerated type: only me, friends, friends of friends, everyone), author, and, in the case of annotations, the content to which this resource is annotating as well as the root of the annotation tree.

Open Accessibility Queries via Views. To support open accessibility queries, a view (`V_Access`) can be created to relate users to resources that the former can access (Fig. 2). Such a view allows us to query the set of resources that a given user may access. The view is the union of four different views, one for each of the four modes of access (i.e., only me, friends, friends of friends, everyone). Each of the four views relates users to resources with policies granting the corresponding mode of access.

Fig. 3 shows stored procedures for retrieving those annotations of a given resource that are accessible to a given user: one procedure for simple annotations, and another for higher order annotations. The reason for using stored procedures instead of inline queries is to optimize the execution time.

```
CREATE VIEW View_Friends AS
SELECT f1.UserID1, f1.UserID2 FROM Friends AS f1
UNION ALL
SELECT f2.UserID2, f2.UserID1 FROM Friends AS f2

CREATE VIEW V_Owner_Acc AS
SELECT RootID, ParentID, ResourceID, OwnerID AS UserID
FROM Resources

CREATE VIEW V_Friends_Acc AS
SELECT r.RootID, r.ParentID, r.ResourceID, v.UserID2 AS UserID
FROM Resources AS r
  JOIN View_Friends AS v ON r.OwnerID = v.UserID1
WHERE (r.PolicyID = 1) OR (r.PolicyID = 2)

CREATE VIEW V_FOF_Acc AS
SELECT r.RootID, r.ParentID, r.ResourceID, F2.UserID2 AS UserID
FROM Resources AS r
  JOIN View_Friends AS F1 ON r.OwnerID = F1.UserID1
    JOIN View_Friends AS F2 ON F1.UserID2 = F2.UserID1
WHERE (r.PolicyID = 2)

CREATE VIEW V_Everyone_Acc AS
SELECT r.RootID, r.ParentID, r.ResourceID, u.UserID
FROM Resources AS r
  CROSS JOIN Users AS u
WHERE (r.PolicyID = 3)

CREATE VIEW V_Access AS
SELECT * FROM V_Owner_Acc UNION SELECT * FROM V_Friends_Acc
  UNION SELECT * FROM V_FOF_Acc UNION SELECT * FROM V_Everyone_Acc
```

Fig. 2. View definitions for reverse accessibility check.

7 Performance Evaluation

This section demonstrates that the performance of the implementation strategy as proposed in the last section has reasonable performance.

7.1 Dataset

Social Network Data. We used an anonymized social network dataset (4,847,571 nodes and 68,993,773 edges) created from LiveJournal [1, 11] and hosted by Stanford Large Network Dataset Collection. LiveJournal is a social network with estimated 10 to 100 millions of users. Users can have blogs and add one another as friends. Friendship in LiveJournal is directed. We therefore created a database view to represent the symmetric closure of the friendship relation, thereby making friendship symmetric as in Facebook.

Resources Data. We generate the resources that are being protected by access control. Each resource has a privacy policy chosen uniformly at random from $\{0, 1, 2, 3\}$, where 0, 1, 2 and 3 correspond to *Only Me*, *Friends*, *Friends of Friends* and *Everyone* respectively. These are the default privacy policies available in Facebook. Each resource's owner is selected uniformly at random from the set of all users in the dataset. Each resource can be either a content or an

```
CREATE PROCEDURE SP_Can_Access @UserID INT, @ResourceID INT AS
BEGIN
SELECT r.ResourceID FROM
 ( SELECT va.ResourceID FROM View_Access AS va
   WHERE (va.UserID = @UserID) AND (va.ResourceID = @ResourceID) )t
    JOIN Resources AS r ON r.ParentID = t.ResourceID
      JOIN View_Access AS va ON va.ResourceID = r.ResourceID
WHERE va.UserID = @UserID
END

CREATE PROCEDURE SP_Recursive_Can_Access @UserID INT, @ResourceID INT AS
BEGIN
WITH Recursive_Can_Access (ResourceID, Level) AS
 (SELECT ResourceID, 0 AS LEVEL
  FROM View_Access
  WHERE ResourceID = @ResourceID AND UserID = @UserID
  UNION ALL
  SELECT va.ResourceID, LEVEL + 1
  FROM View_Access AS va
    JOIN Recursive_Can_Access AS r ON va.ParentID = r.ResourceID
  WHERE va.UserID = @UserID AND va.RootID = @ResourceID)
SELECT ResourceID, LEVEL
FROM Recursive_Can_Access
END
```

Fig. 3. Procedures for retrieving simple and higher order annotations.

annotation. Denote by R the set of all resources, C the set of (non-annotation) contents, and A the set of annotations. We have: $R = C \uplus A$. We design two separate configurations of experiments, one for simple annotations and another for higher order annotations. As a result, the way we relate annotations to contents differ for each configuration. Below we show how we relate annotations to contents in each configuration.

The Simple Annotations Configuration. We randomly generate a function $f : A \to C$ to assign annotations to contents. The generation of f is controlled by a parameter $\mathsf{ratio} = \left\lfloor \frac{|A|}{|C|} \right\rfloor$, which is the average number of annotations that each content has. Function f maps each element in domain A to an element in codomain C uniformly at random with probability $\frac{1}{|C|}$.

The Higher Order Annotations Configuration. We randomly generate a function $f : A \to R$ to map annotations to resources. We generation of f is again controlled by the parameter $\mathsf{ratio} = \left\lfloor \frac{|A|}{|C|} \right\rfloor$. The constraint which function f must satisfy is that annotation assignment must result in a *forest* (disjoint union of trees) over R (no circles).

7.2 Setup

The experiments are conducted both on a consumer-scale machine (aka local) and the Microsoft cloud called Windows Azure (aka Azure). The results from

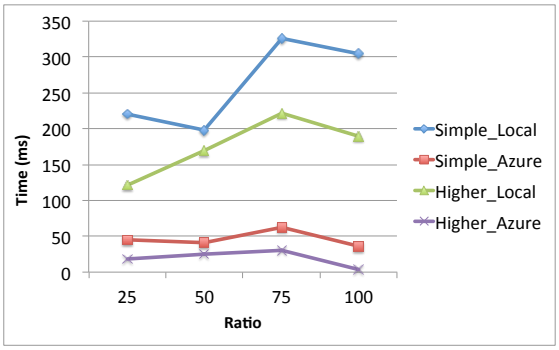

Fig. 4. Performance figures for the two experimental configurations, each repeated in both the local and Azure environment.

both environments are reported. We repeat the experiments in two characteristically different computing environments to give an idea of the range of performance that one can expect in reality. The consumer-scale machine is but a notebook computer, and thus it represents the pessimistic lower bound of performance. The Azure environment is likely more representative of the kind of server-side capability that a social computing vendor possesses.

The local machine is equipped with Microsoft SQL Server (2005 and 2012) and 64-bit Windows 7 Professional (Service Pack 1), and has the following hardware configuration: 2.4 GHz Intel Core 2 Duo Processor, 4 GB 1067 MHz DDR3 Memory, SATA disk. The database is later migrated to Windows Azure, and experiments are conducted on the SQL server provided in Windows Azure. The database is hosted on a Windows Azure server located in north central USA. Most of the experiments on Azure are run around midnight when we conjecture there is less load on the server.

7.3 Measurements

For each of the two configurations (simple annotations and higher order annotations), we measure the execution time of submitting the corresponding query in Fig. 3 to the SQL Server. The measurement is repeated for 1000 times, each with randomly chosen arguments (repetitive arguments are avoided), and the average execution time is computed. Each query receives two arguments: (a) a user of the social network and (b) a content. The users are chosen in such a way that they have access to the content itself according to that content's access control policy. The result set for these queries are the annotations of that content for which the user is allowed to access.

7.4 Results and Interpretation

Fig. 4 shows the response time for retrieving accessible annotations. Note that, when a consumer-scale machine (local) is used, the retrieval time is between 100−

−350 milliseconds. When a server-scale machine (Azure) is used, the retrieval time is no higher than 70 milliseconds.

According to [14, Chapter 5] and [13], the following response time limits must be considered in interactive applications. (1) 0.1 second is the limit for the users to feel that system works instantaneously and that they are directly performing the manipulation (typing in a text box and viewing the text typed simultaneously). (2) 1.0 second is the limit for the users to keep their flow of thoughts and they feel they are freely navigating (although they feel the delay, they also feel the computer is working). (3) 10 seconds is the limit to keep users attention. Taking into consideration network latency ($30 - 300$ milliseconds for web sites such as Facebook [3]), and using server-scale machines like Windows Azure, response time remains in the acceptable range of 0.1 to 1.0 second.

8 Discussions and Future Work

Of the five classical examples of multiple stakeholders (i.e., tagging, commenting, sharing, foreign contents, and friendship articulation), the design patterns proposed in this work can address three of them with very simple designs of access control. The core observation that we depend on is Design Principle 2, which asserts that components of composite objects should be protected by separate access control policies when the stakeholders of the components are different.

The above implies that there are two classes of multiple-stakeholder scenarios that cannot be addressed in the manners suggested in this work.

1. ***Joint Assertions***. Some contents are atomic: they are not made up of separately identifiable components. An example is the assertion of a relationship between two parties. Such contents are created under the consent of multiple parties (e.g., befriending requires the consent of the two friends), and thus multiple stakeholders are involved.
2. ***Collaborative Authoring***. Some contents, such as wiki pages, are composite, but each must be taken as a whole. These contents are results of collaboration among multiple authors, and yet authorship is attributed to the entire product and cannot be attributed to the parts. Thus the finished product affects the privacy of multiple stakeholders.

In these 2 cases, general-purpose schemes for multiple stakeholders [7–9, 15–17, 21] are absolutely irreplaceable. Future work in the study of protection schemes for multiple stakeholders should focus on the above two classes of scenarios.

Acknowledgments This work is supported in part by an NSERC Discovery Grant and a Canada Research Chair.

References

1. L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group formation in large social networks: Membership, growth, and evolution. In *Proceedings of KDD'06*, pages 44–54, Philadelphia, PA, USA, 2006. ACM.

2. K. J. Biba. Integrity considerations for secure computer systems. Technical Report ESD-TR-76-372, Electronic Systems Division, Air Force Systems Command, United States Air Force, Apr. 1977.
3. CityCloud. Some interesting bits about latency. `https://www.citycloud.com/city-cloud/some-interesting-bits-about-latency/`, Aug. 2012.
4. Facebook Help Center. `https://www.facebook.com/help/www/255898821192992`, Aug. 2013.
5. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison Wesley, 1994.
6. G. S. Graham and P. J. Denning. Protection: Principles and practice. In *Proceedings of the 1972 AFIPS Spring Joint Computer Conference*, volume 40, pages 417–429, Atlantic City, New Jersey, USA, May 1972.
7. H. Hu and G.-J. Ahn. Multiparty authorization framework for data sharing in online social networks. In *Proceedings of DBSec'11*, pages 29–43, Richmond, VA, USA, 2011.
8. H. Hu, G.-J. Ahn, and J. Jorgensen. Detecting and resolving privacy conflicts for collaborative data sharing in online social networks. In *Proceedings of ACSAC'11*, pages 103–112, Orlando, Florida, USA, 2011.
9. H. Hu, G.-J. Ahn, and J. Jorgensen. Multiparty access control for online social networks: Model and mechanisms. *IEEE Transactions on Knowledge and Data Engineering*, 2013.
10. M. Kumar. How Facebook Ticker Exposing Your Information and Behavior Without Your Knowledge. `http://thehackernews.com/2011/10/how-facebook-ticker-exposing-your.html`, October 2011.
11. J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.
12. N. Li and M. V. Tripunitara. On safety in discretionary access control. In *Proceedings of IEEE S&P'05*, pages 96–109, Oakland, CA, USA, May 2005.
13. R. B. Miller. Response time in man-computer conversational transactions. In *Proceedings of the 1968 AFIPS Fall Joint Computer Conference, Part I*, volume 33, pages 267–277, San Francisco, CA, USA, Dec. 1968.
14. J. Nielsen. *Usability Engineering*. Morgan Kaufmann, 1993.
15. A. C. Squicciarini, M. Shehab, and F. Paci. Collective privacy management in social networks. In *Proceedings of WWW'09*, pages 521–530, Madrid, Spain, 2009.
16. A. C. Squicciarini, M. Shehab, and J. Wede. Privacy policies for shared content in social network sites. *The VLDB Journal*, 19(6):777–796, Dec. 2010.
17. A. C. Squicciarini, H. Xu, and X. L. Zhang. CoPE : Enabling collaborative privacy management in online social networks. *Journal of the American Society for Information Science*, 62(3):521–534, 2011.
18. The Social CMO. New Facebook Ticker Is Invasion of Privacy. `http://www.thesocialcmo.com/blog/2011/09/new-facebook-ticker-is-invasion-of-privacy/`, Sept. 2011.
19. K. Thomas, C. Grier, and D. M. Nicol. Unfriendly: Multi-party privacy risks in social networks. In *Proceedings of PETS'10*, pages 236–252, Berlin, Germany, 2010.
20. C. Washbrook. Facebook's Ticker Privacy Scare, and What You Should Do About It. `http://nakedsecurity.sophos.com/2011/09/26/facebook-ticker-privacy-scare/`, Sept. 2011.
21. R. Wishart, D. Corapi, S. Marinovic, and M. Sloman. Collaborative privacy policy authoring in a social networking context. In *Proceedings of IEEE POLICY'10*, pages 1–8, Fairfax, VA, USA, 2010.