A Logic of Creation in Online Social Networks

Brian Whitworth[1], Lech Janczewski[2] and Adnan Ahmad[1]
[1]Massey University, Auckland, New Zealand
[2] The University of Auckland, New Zealand

Abstract - *A community is a social entity that by norms, laws or ethics grants its citizens rights - social permissions to act. It does so to help itself, as a community that prospers helps its members. Online social networks are computer based communities whose social requirements are not too different from any other. Access control in these networks requires some logical foundation to build upon. Without an agreed logical basis to distribute social rights, current access control models are based on intuition, experience or trial and error. This paper suggests anonine entity creation logic based on the socio-technical approach – use the knowledge of physical society as the basis of information rights model for online communities. Social axioms give a theoretical base for rights analysis that could not only satisfy technical requirements like efficiency but also social requirements like fairness.*

Keywords: social computing, socio-technical, access control, security, social network

1 Introduction

The last decade has seen extreme multi-user systems emerge – *online social networks* (OSN) where millions of users share billions of resources and grant each other access rights (Carminati, 2009). A social network is a type of socio-technical system (STS), which is a social system operating upon a technical base (Whitworth, 2009), e.g. wikis, social media, e-trade or chat. Every STS has both social and technical requirements, so can fail by social or technical error, e.g. by allocating permissions unfairly or inefficiently.

Online access to resources and information is managed through an access control system (ACS), which restricts who can access what based on a permission matrix which for friend interactions increases geometrically not linearly with group size. So for hundreds of millions of people, the possible connections are astronomical. Each account also adds hundreds or thousands of photos or comments a year and each user wants the sort of control over their domain previously reserved only for system administrators. With the world population at seven billion and growing, if Facebook's current 800 million active accounts is just the beginning, matrix access methods may be ending their useful life.

In traditional ACS, access is granted to predefined users, but OSN profiles can be created by users new to the system and rights allocation is over constantly new objects. Currently, access control in social networks is based on designer intuition, experience or even trial and error, with no agreed common base. The base proposed here is social requirements. As social networks are here to stay and growing in number and size, a common model of distributed rights allocation can identify socio-technical design *patterns* (Alexander, 1964).

Privacy is one OSN social requirement, as connecting to others raises privacy concerns (Simpson, 2008). People want to contribute personal stuff to online social networks without worrying about its unauthorized disclosure (Ahmad and Whitworth, 2011). Another is Locke's idea that one should own what one creates, whether a book, a painting or an online photo (Locke, 1975). If so, everything posted on an OSN should be owned, and conversely if people own their posts, they should manage their access control. Access control in OSNs today is more about access than control because people want to share as well as keep private. Essentially, if people don't own the resources they contribute, why bother to add them at all? Why do work for someone else to get the result? If people don't contribute to an STS there is no user community and it fails socially. Ownership of newly created online objects is critical to OSN success for social reasons.

The aim is a system that works both in technical practice (efficient, consistent and reachable) and in social practice (fair, productive and understandable). The access control logic outlined here could generalize to any socio-technical system.

2 Requirements

A socio-technical system is a social system on a technical base, as a socio-physical system is a social system on a physical base. Socio-technical design (Mumford, 1995; Porra and Hirscheim, 2007) involves technical *and* social requirements, to model not just what *can* be done but what *should* be done. Social requirments, usually applied to workplace management, are here applied to software design.

Social synergy is people working together to increase each other's outcomes (it isn't just people adding efforts, say to lift a heavy log together). Communities that enable positive synergy will attract citizens while negative synergy will repel them, e.g. the Berlin wall was a short term physical attempt to block a long term social synergy effect. People move to societies where synergy increases productivity. The same occurs online but much faster, as a Facebook or Gmail "citizen" need only set up a new identity to "emigrate". A few clicks is all it takes to leave one application "country" for another. Social networks ignore social requirements at their peril, as without a citizen base how can they survive?

An access control system needs to satisfy:

1. *Technical requirements.*:
   a. *Efficiency.* To reduce complexity to support use factors like response delay (Ahmad and Whitworth, 2011).
   b. *Consistency.* To compile, code must be logical.
   c. *Reachability*. To not hang, the logic must be reachable.
2. *Social requirements.*
   a. *Ownership.* To reduce resource conflict and allow trade.
   b. *Freedom.* To increase participation we must be free.
   c. *Fairness.* Fairness is accountability for ones acts on others. The goal of justice systems is just that, not equity (Rawls, 2001). People tend to avoid unfair situations and even prefer fairness to personal benefit (Adams, 1965).
   d. *Privacy*. Without control over how it is seen, one will not post personal information online.
   e. *Transparency.* Is a citizen's right to know the social rules affecting them. Physical societies make laws public, e.g. road rules, so online access control systems should be the same.

Link operations (Whitworth and Bieber, 2002), local administration (Ahmad and Whitworth, 2011) and rights reallocation (Ahmad *et al.* 2012) are discussed elsewhere. This paper now applies the socio-technical approach to the logic of online creation.

3 A rights analysis

Access control in traditional computing involves three basic entities, i.e. subjects, objects and rights. In online communities the entities remain the same but their semantic and interpretation changes based on the social interaction.

3.1 Social Actors

A persona is an information entity *representing* a person, e.g. an avatar, logon profile, WebMail account, wall or channel. An online social system thus constitutes social actors who interact via personae (Whitworth *et al.* 2006). This paper prefers the term “actors” for social interactions, rather than “users”, for several reasons. First, a user is characterized in relationship to a particular application, whereas a social actor can participate in many application setting (Kling *et al.* 2003). Second, the term “user” generally implies a single relationship to a system while social actors have multiple relationships (Kling *et al.* 2003). Finally, given multiple relationships in multiple settings, an actor may have conflicting or ambiguous demands that require a choice over the actions they perform (Lamb and Kling, 2003).

3.1.1 Who owns the persona?

In the physical world, freedom is the right to control one's body, to not be a slave to another. If freedom online is the right to control one's online body, or persona, one should be able to edit or delete it, yet many systems don't permit this (Lessig, 1999). A system offers freedom if an actor can remove themself from it, e.g. delete a Facebook wall or YouTube channel with nothing left behind. The social logic is that one *owns* oneself online, i.e. an online persona does *not* belong to a system administrator . If someone else can control my persona, I am an online slave. As society can imprison criminals, freedom is a granted right, i.e. a privilege, but modern societies grant lawful citizens freedom, giving the access control operational principle:

P1. A persona should be owned by itself.

Display grants another the right to view an entity. Privacy, a persona's right to control its display, gives the access control operational principle:

P2. Displaying a persona requires its consent.

This social pattern is also general, e.g. in a Facebook or Linkedin registration creates a persona but displaying it to other registrants is by consent. As the SA owns the public list, to put a persona on a public view list needs the permission of both its owner and the list owner, jointly allocated. Table 1 summarizes these persona access rights.

TABLE 1. PERSONA ACCESS RIGHTS.

Persona	*View*	*Delete*	*Edit*	*Display*	*Ban*	*Allocate*
SA	√			½√	√	
Owner	√	√	√	½√		√

3.2 Objects

Objects as passive entities are subject to operations. They convey meaning, i.e. evoke cognitive processing, e.g. a photo. Objects can be of two types, items and spaces.

3.2.1 Items

An item is a simple object with no dependents, like a board post. It can be deleted, edited or viewed. If the system object hierarchy were a tree, its leaves would be items. Items can be of different types, e.g.:

1. *Comment*: An item whose meaning depends on another, e.g. "*I agree"* makes no sense without a source item.
2. *Message*: An item with sender/receiver, e.g. an email.

3. *Vote:* An item that conveys a choice position to a response set.

3.2.2 Spaces

As leaves need branches so items need spaces to carry them, e.g. an online wall that accepts photos or notes. In information terms, a *space* is a complex object with dependant entities. It can be deleted, edited or viewed as an item can, but can also can contain other objects, e.g. a bulletin board. A space is a *parent* to any *child* entities it contains, as they depend on it to exist. Deleting a space deletes its contents for that reason, e.g. deleting a board deletes its posts. The *move* operation changes the parent space of an object. Allowing spaces improves access efficiency, e.g. one can deny access to every object in a space by denying entry to the space, giving the principle:

P3: Every entity has a parent space, up to the system space.

An access control system can assume that every entity has a parent space (except for the system itself). Its *ancestors* are the set of all spaces that contain it, up to the system itself, as the first ancestor. Equally the *offspring* of a space are any child objects it contains and any other derived children.

3.3 The information system as an entity

In this model, the information system itself is an entity that can be acted upon by its owner, or SA, e.g. Wikipedia has Jimmy Wales. While control systems make decisions, and decision support systems recommend them, access control systems merely specify allowed acts, i.e. give choices. Ownership requires that all use rights be set, giving the access control operational principle:

P4. All entity use rights must be allocated.

Unless all use rights are allocated, an access control system has no basis to operate.

3.4 Operations

Operations are actor initiated methods that target information entities subject to access control. Passive operations are null acts that don't change their target, e.g. view. To be accountable for an object one must be able to see it, i.e. use rights imply view rights, giving the access control operational principle:

P5: Any right to use an object implies a right to view it.

Communicative acts like email involve sender and receiver actors, i.e. two parties. Social communication is considered a joint act either party can negate, e.g. "*Can I talk to you*?" is asking permission to communicate. Legitimate communication first opens a mutual consent *channel* then sends messages (Whitworth and Liu, 2009). Communication fairness gives the access control operational principle:

P6: A communication act requires mutual consent.

Email is losing ground to systems that respect communication mutuality for this reason. Its technical design ignored social needs, and so despite the best technical efforts of filters it is slowly being consumed by spam and giving way to more legitimate communication forms (Whitworth and Liu, 2009). Socio-technical systems are only socially sustainable if they are *legitimate by design* (Whitworth and deMoor, 2003).

3.5 Rights

Online rights management is about defining legitimate choices – individuals still choose what they do. Access control defines what online actors *can* do not what they *must* do. A right is a system permission for actor *(A)* to apply operation *(O)* to entity *(E)*, or in formal terms:

$$Right = (Actor, Entity, Operation) = (A, E, O)\ldots (i)$$

The actor is any social entity, e.g. a persona. The entity can be an object, a social entity or a right, so a persona can act on itself. The operation is any that is available to that entity.

3.5.1 Roles

Roles like parent, friend or boss are used in norms and laws to simplify rights management, e.g. owner as the generic party with the right to use an owned object. In online access control, roles both simplify rights management and improve actor acceptability. Roles are loosely seen as an actor set, but here are an actor set in a rights statement, e.g. the friend role is a set of people in the context of stated permissions. So roles are here generic rights, giving the access control principle:

P7: A role is an entity right expressed in general terms.

Roles are the variable statements of social logic, e.g. the *owner* role is:

$$Role_{Owner} = (\mathbf{Actor}, Entity_{Any}, Operation_{All})\ldots(ii)$$

Setting an actor to own Entity$_{Any}$ is to allocate the unspecified **Actor** pointer to them. Roles reduce right management complexity and are flexible enough to accommodate social variety, e.g. the friend role lets one add or remove the others who can view photos posted on a social network wall:

$$Role_{Friend} = (\mathbf{Actor}, Entity_{Wall}, Operation_{View})\ldots (iii)$$

To friend another adds them to a role actor set and to unfriend removes them. To "friend" doesn't change the target persona but the actor's role, so it is really an act upon a role. This gives the operational principle:

P8. A space owner can ban a persona without their consent.

3.5.2 Meta-rights

If a right is an information entity, it can also be acted on. Operations upon rights (as opposed to entities) are here called *allocations*. Changing rights implies rights to change rights i.e. *meta-rights*. A meta-right is the right to re-allocate a right, e.g. a tenant renting an apartment has use rights for a time but the landlord owner keeps the meta-rights. For practical reasons, a meta-right is also classed as a right, giving the access control operational principle:

P9. A meta-right is the right to allocate any entity right, including itself.

In formal terms:

$$Right_{MetaRight} = R\ (Actor,\ Right,\ Operation_{Allocate}) \ldots (iv)$$

where the entity acted on is a right.

4 A logic of creation

To create an information object from nothing is as impossible in an online space as it is in a physical one. Creation *cannot* be an act upon the object created, as it by definition doesn't exist before it is created. Likewise, an actor can't request an access control permission to create for an object that doesn't exist yet. Also, to create an information object it's attribute structure must already be known, i.e. exist within the system. To be consistent, creation is an act upon the system, or in general, an act on the space containing the created object. This gives the operational principle:

P10. Creation is an act on a space, up to the system space.

It is well defined as a system always has a space, as the system itself is the first space. Creating is also an act upon a space because it changes the space, as it as now contains the created object. If creation is always an act upon a space, it follows that the right to create in a space belongs to the space owner:

$$Right_{Create} = R\ (Owner_{Space},\ Space\ ,\ Operation_{Create}) \ldots\ (v)$$

This allows an access control system to be initialized with a system administrator owning a system space with all rights, including create rights, that then evolves into a community as the SA give rights away. To create a community of others, one must give rights away (Gaaloul, Schaad, and Flegel, 2008).

The logic can generalize to any space - the right to create in the space is initially allocated to the space owner who can allocate it to others who enter the space. So to create a board post, YouTube video, blog comment or conference paper requires the board, video, blog or conference owner's permission. How create rights are allocated is given in more detail later, but space owners can vary (Lessig, 1999):

1. *Object type.* The space owner may limit object type, e.g. in a conference, e.g. the right to create paper in a track isn't the right to create a mini-track.
2. *Operations*. A comment isn't usually editable once posted but ArXiv lets authors edit publications.
3. *Exclusivity*. Journals give authors exclusive edit rights while Wikipedia lets anyone edit any creation.
4. *Visibility*. Bulletin boards let you see what others submit but conferences don't until the review phase is done.
5. *Defaults*. Space owners set created entity default values.

In all the above, transparency is the right to know the creation "deal" in advance. A space owner can delegate all, none, or some of their creation rights, but receivers also have the right to know what they are getting, giving the access control operational principle:

P11. An actor can view any rights that could apply to them .

Successful socio-technical systems like Facebook, YouTube and Wikipedia, all do this.

4.1 Creator Ownership

Object creation is a simple technical act but a complex social one, as a newly created entity's rights are initially unallocated. Locke argued that creators owning their creations is fair and increases prosperity, whether it is a farmer's crop, a painter's painting or a hunter's catch (Locke, 1975). A community that grants producers the right to their products produces more, while there is no incentive to create by effort for others to own. This gives the access control principle:

P12. The creator of new entity should immediately gain all rights to it.

This conveniently resolves the issue of how to allocate the rights to newly created object - they are allocated to its creator, including meta-rights. Create then immediately gives the right to edit, which is useful as create sets no new object values. Yet P12 isn't what must happen - a technical program can create an information object however it likes, e.g. give its ownership to the system administrator as in traditional applications. Creator ownership is a requirement for social success not a technical necessity.

4.2 Role Allocations

When an entity is created the following roles can be assigned:

1. *Owner role:* Given meta-rights to the entity.
2. *Ancestor role:* As a created entity becomes part of the space it is created in, it should be visible to its parent space owner who is accountable for their space. Fairness entitles a space owner to view any creation change they allowed. Privacy does not contradict this, as is the right to limit the display of personal attributes, not of objects owned. By the same logic, an entity should be visible to all its ancestors, giving the operational principle:

   P13. A space owner should have the right to view any offspring.

   with the ancestor role:

   $$Role_{Ancestor} = (AncestorOwner\ ,Entity\ ,\ View) \ldots (vi)$$

   A posted conference paper could be visible to its conference, track and mini-track chairs, but not to other track or mini-track chairs. Ancestors may receive *notifications* of new additions.

3. *Offspring role*: Consistency requires that an entity has already entered its parent space, so can view whatever is displayed in it, i.e. children can always see the space they are in. By extension, they can also enter any ancestor space as they are already in it, giving the access control operational principle:

   P14. An entity owner has the right to enter its ancestors.

   e.g. adding a paper to a mini-track should let one enter the mini-track, track and conference spaces to view whatever is displayed there. The offspring role is:

   $$Role_{Offspring} = (OffspringOwner, Space, Enter) \ldots (vii)$$

4. *Local public role*: A space owner can create and own a local public role, to define what others can see or do in the space:

   $$Role_{LocalPublic} = (LocalPublic, Space, Operation_{Any}) \ldots (viii)$$

   Actors in a local public role can be set manually, as friends are allocated, or set to a *general public* list given by the system.

5. *Sibling role*: Owners of other entities in the same space, *may* get view rights, e.g. that one can only view items in a space after adding one oneself, i.e. visibility is allocated to *siblings*. However authors who submit conference papers get no such right to view siblings, i.e. see what others have submitted to the same track.

A space owner can grant any right they own to their local public subject to conditions, e.g. a Wikipedia create condition is to allow public edits.

4.3 The create process

Technically, creating an entity is simple – a program just creates it, but socially adding to another's space is not a one-step act, e.g. adding a YouTube video involves:

1. *Registration*. Creating a YouTube public role persona.
2. *Entry*. YouTube allows public entry, if not banned.
3. *Creation*. YouTube lets the public role upload videos.
4. *Edit*. One gets edit right to title, notes and properties.
5. *Submit*. To display to the public view.
6. *Display*. The space displays it so the public sees it.

In this model, YouTube gives create video rights to anyone who has registered in the public role (1). They enter the YouTube space (2) and create a video by uploading or recording, which they own (3). They can then in private view it and edit details (4). At this point, the video is visible to themselves and administrators but not to the public, and they can still delete it. The video is then submitted to YouTube for display to its public (5), which usually occurs quickly as YouTube delegates display rights (6). Note that to create, edit and display a video are distinct steps. As YouTube only delegated display rights, it can still reject videos that fail its copyright or decency rules by un-delegation. This rejection is not a delete, as the video owner can still view, edit and resubmit it.

In contrast, a purely technology based rights allocation might let space owners delete items at will. Ignoring creator ownership would discourage social participation and the system might fail socially. Modeling the display of an object in a space as a social transaction between its creator and the space owner allows the sharing of control in many ways, from laissez-faire to dictatorial control. What works can then be decided by the social outcome.

The above logic generalizes easily, e.g. a YouTube video is itself a space with dependent comments and votes. YouTube is consistent and fair as the same principles apply as the video creator becomes a space owner. They can choose to allow comments or votes on their video, i.e. they can grant rights to their domain space, just as Facebook citizens do. Socio-technical systems succeed by allocating social rights legitimately.

Rights logic is powerful but complex, as people can form groups, objects can contain other objects and rights can overlap and contradict, e.g. free speech is not the right to defame. A socio-technical designer might wonder, if even legal theorists can't agree on all social rights, how can we cope? Yet some justice is always better than none, whether online or off. To do nothing until perfect justice is defined is not how social evolution occurs.

5 Theoretical analysis

The efficiency, reachability and consistency of the proposed logic is analyzed in this section.

5.1 Efficiency

In OSN, access control models face a serious scalability problem, as potentially many more subjects must be mapped to many more resources, regardless of whether a subject has access rights over a resource or not. We can use $u \times o \times r$ matrix *MAT* to estimate the relations between users, objects and permissions, where u is the number of users, o is the number of objects and r is the number of access permissions. The authorization matrix

$$|MAT| = subject \times object \times access \ldots (ix)$$

is therefore huge and diverse (Kerschbaum, 2010). One often proposed answer is role-based access control (RBAC) (Sandhu *et al.*, 1996), which succeeds in reducing complexity (Lee *et al.*, 2006; Wu *et al.*, 2008) by dividing the authorization matrix using a level of indirection via the role concept:

$$subject \times role; |MAT| = role \times object \times access \ldots (x)$$

However for an OSN with millions of users, the number of access control entries still remains a bottle neck, even with RBAC, e.g. currently Facebook reports over 750 million active users with 90 resources added by each every month[1]. In a traditional DAC access control model, this is over 151 trillion access control entries per month, where every request must traverse the whole list. And the condition with RBAC is even worse as implementing ownership with RBAC is more expensive (Sandhu and Munawer, 1998). The proposed model reduces the authorization matrix, as now the visibility of objects is not across the whole system but limited to the social circle of each social owner. This limits actors having potential access over resources:

$$\text{Potential Users} = \text{Subject} \cap \text{Social Circle} \ldots (xi)$$

The authorization matrix for objects of one owner then reduces by

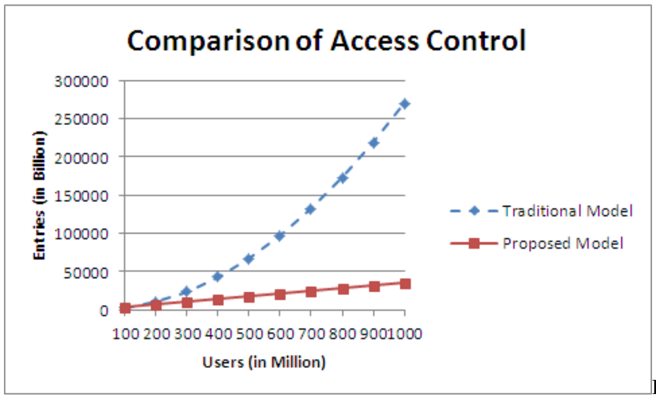
$$\text{Potential Users} \times \text{Local Roles; Objects} \times \text{Owner} \ldots (xii)$$

And the authorization matrix for the whole system under the proposed model, using local roles *(LR)* and permissions for a space is given by

$$|MAT| = \sum_{i=1}^{N} \left| \sum_{j=1}^{n} LR_j \times \sum_{j=1}^{n} Obj_{Owner} \times \sum_{j=1}^{n} Permissions_{Space} \right| \ldots (xiii)$$

where *N* is number of users present in the whole system and *n* is the number of *local roles*, *objects* and *permissions* present in the *Owner Space.*

These settings give the proposed model fewer access control entries. In the above case, the number of entries is reduced from 151 trillion to 25 trillion. Figure 1 shows the number of access control entries generated by user number for a fixed object contribution in the two cases.



Fi

Figure 1. Access control matrix magnitude for different models

5.2 Reachability

For reachability, suppose that Alice creates one object *o1* in David *space* and one object *o2* in her own *space*. She then assign Greg and Frank the edit rights over *o1* and *o2*, and assign Eric, Carl and Bob the view rights over *o1* and *o2*. Further suppose that Bob creates a child object *o3* dependent on *o2* and gives its view rights to Carl. The access control model instance depicting this scenario can be seen in Figure 2.

Reachability (Bertino, 2003) is the ability to determine whether a certain authorization can raise the condition of conflict with another authorization in the system. It is the state which can occur when one authorization can lead to another and the second authorization is not valid under the current state. It can be categorized into three classes: *a)* a negative authorization can be derived from a positive authorization, *b)* a positive authorization can be derived from a negative authorization, and *c)* a positive authorization can be derived from a positive authorization. First and second reachability type can cause conflict in the system and shows that the model is not stable under the current state. Third type does not offer any serious concern but gives insights about the behavior of any authorization grant. Reachability is a mean to determine the effect of granting an authorization and can assist the system administrators to specify the authorization in a well-defined manner.

Formally, reachability can be defined as a condition if an authorization *Auth* $(X_1, X_2 \ldots X_n, +)$ can be reached from another authorization *Auth* $(Y_1, Y_2 \ldots Y_n, -)$, and both of the authorizations are not in the same authorization scheme. The above three classes can be reduced to the Boolean formula whether a) *Auth (S: U/ UR/ GR, O :o,* $R{:}R_i$ *, ϒ:+)* ← *Auth (S: U/ UR/ GR, O :o,* $R{:}R_j$ *, ϒ:-)*, b) *Auth (S: U/ UR/ GR, O :o,* $R{:}R_i$ *, ϒ:+)* → *Auth (S: U/ UR/ GR, O :o,* $R{:}R_j$ *, ϒ:-)*, and c) *Auth (S: U/ UR/ GR, O :o,* $R{:}R_i$ *, ϒ:+)* ← *Auth (S: U/ UR/ GR, O :o,* $R{:}R_j$ *, ϒ:+).*

Following the model and its authorization state presented in figure 2, some conclusions can be drawn.

It is clear from constraints *#c3* and *#c1* that *(U: #8, O : #10, R: #12, ϒ: -)* cannot be reached from *(U: #8, O : #10, R: #11, ϒ:+).*

Using the constraint *#c2*, the derivation tree of *Auth (S : #ar4 ,O : #10, R : #13, ϒ: +)* contains *Auth(U: #4, O : #10, R: #11, ϒ: +)*, so *Auth(U: #4, O : #10, R: #11, ϒ: -)* cannot be reached using the same authorization tree.

The *Auth (U: #6, O : #10, R: #11, ϒ:+)* can be reached from the derivation tree of *Auth (S : #ar3 ,O : #10, R : #12, ϒ: +)* using constraints *#c3* and *#c1*.

5.3 Consistency

An access control model is considered consistent (Bertino, 2003) if there is at least one instance of that model satisfying all the specified constraints. Consistency is useful to analyze the model in terms of authorization granted to particular user and its allocation in various roles. Consistency of a model can be accessed *a)* if some role *A* is higher than role *B* in hierarchy, then the rights allocation to *B* is a subset of rights allocation to *A*, the same rule can be stated as if higher rights are allowed then lower rights are also allowed to a user, *b)* by not authorizing lower rights to role *A* means denying of higher rights to the same role, *c)* after the reallocation of some rights, the new recipient and the old holder of rights should not have the same set of rights.

[1] Facebook statistics, *http://www.facebook.com/press/info.php?statistics* .

Formally, an access control model is considered consistent if an instance I of that model satisfies $I \rightarrow Constrainsts$. The consistency problem can be reduced to satisfy a) *Auth (R, O,* $X_2 \ldots X_n$*, +)* $\in$ *Auth (R', O',* $X'_2 \ldots X'_n$*, +) iff R >R',* b) *iff Auth (*$X_1, X_2 \ldots X_n$*, +) > Auth (*$X'_1, X'_2 \ldots X'_n$*, +)* then *not_Auth (*$X'_1, X'_2 \ldots X'_n$*, +)* $\rightarrow$ *not_Auth (*$X_1, X_2 \ldots X_n$*, +)* and c) *Auth (Old,* $X_2 \ldots X_n$*, +)* $\neq$ *Auth (New,* $X'_2 \ldots X'_n$*, +).*
Following figure 2, *Auth (U: #6, O : #10, R: #11, ϒ:+)* is a subset of *(U: #6, O : #10, R: #12, ϒ:+)*, as *R:#11* is a subset of *R:#12*, and *(U: #6, O : #10, R: #12, ϒ:+)* is a subset of *(U: #6, O : #10, R: #12, ϒ:+)*, as *R:#12* is a subset of *R :#13*.
The model in figure 2 is static depicting only one scenario, but its dynamic nature for supporting the rights re-allocation can be seen from *Auth (S : Old, O: o, R:X) < Auth (S : New, O: o, R:X'), Auth (S : Owner* $_{Secondary}$*, O: o, R:X) < Auth (S : Owner* $_{Primary}$*, O: o, R:X')* and *Auth (S : Delegator, O: o, R:X) < Auth (S : Delegatee, O: o, R:X').*

6 Conclusion

Online social networks cannot prosper without user participation. If the Internet is to be a global community, it must agree on a consistent logic of online social rights. This paper suggests an access control framework to meet social demands like creator ownership and technical demands like efficiency. This progress is already happening in OSNs, but what is proposed here is not just adding some rights to some code, but an access control module consistently managing social rights within the security kernel.

The next project phase is to integrate ownership logic, distributed control, rights reallocation in an efficient and consistent access control model and trial it as a plugin for an NSF granted open knowledge exchange (OKE) system. It will also generate human readable reports to notify actors of rights, i.e. be transparent. Access control offers a social "road code", to reduce unsustainable social interactions, to increase social trust and synergy, to reduce social errors and conflict, and to reduce community governance overheads. A socio-technical systems must be socially valid as well as technically efficient to sustain over time.

Acknowledgement

This work has been sponsored by National Science Foundation (NSF), USA, under award number 0968445. "OKES: An open knowledge exchange system to promote meta-disciplinary collaboration based on socio-technical principles".

7 References

Adams, J. (1965). Inequity in Social Exchange. In L. Berkowitz, *Advances in Experimental Social Psychology* (pp. 267-299). New York: Ed. Academic Press.

Ahmad, A., & Whitworth, B. (2011). Access Control Taxonomy for Social Networks. *International Conference on information assurance and security, IAS'11.*

Ahmad, A., & Whitworth, B. (2011). Distributed Access Control for Social Networks. *International conference of information assurance and security, IAS'11.*

Ahmad, A., Whitworth, B., & Janczewski, L. (2012). Logic of Rights Reallocation in Social Networks. *IFIP International Information Security and Privacy Conference.*

Ahmad, A., Whitworth, B., & Janczewski, L. (2012). More Choices, More Control: Extending Access Control by Rights Reallocation. Submitted in *International Conference on Trust, Security and Privacy in Computing and Communications*

Alexander, C. (1964). Notes on the Synthesis of Form. *Cambridge: Harvard University Press.*

Bertino, E. and Catania, B. and Ferrari, E. and Perlasca, P. (2003) A logical framework for reasoning about access control models, *ACM Trans. Inf. Syst. Secur. 6 71-127*

Carminati, B., Ferrari, E., Heatherly, R., Kantarcioglu, M. and Thuraisingham, B. M. (2009). A semantic web based framework for social network access control. *SACMAT*, (pp. 177-186).

Gaaloul, K., Flegel, U., & Schaad, A. (2008). A secure task delegation model for workflows. *International Conference on Emerging Security Information, Systems and Technologies.*

Kerschbaum, F. (2010) An access control model for mobile physical objects, *Proceeding of the 15th ACM symposium on Access control models and technologies SACMAT*.

Kling, R., McKim, G., & King, A. (2003). A bit more to it: Scholarly Communication Forums as Socio-Technical Interaction Networks. *Journal of the American Society for Information Science and Technology*, 54(1), 47-67. doi: 10.1002/asi.10154

Lamb, R., & Kling, R. (2003). Reconceptualizing users as social actors in information systems research. *Mis Quarterly,* 27(2), 197-236.

Lee, H. Lee, K. and Chung, M. (20006) Enterprise application framework for constructing secure RFID application, *Proceedings of the 1st International Conference on Hybrid Information Technology*.

Lessig, L. (1999). Code and other laws of cyberspace. *New York: Basic Books.*

Locke, J. (1975). An essay concerning human understanding. *Oxford University Press.*

Mumford, E. (1995). Effective Systems Design and Requirement Analysis, *Information System Series, Palgrave Macmillan*.

Porra, J. and Hirscheim, R. (2007) A lifetime of theory and action on the ethical use of computers. A dialogue with Enid Mumford, *JAIS, vol. 8, no. 9, pp. 467–478.*

Rawls, J. (2001). Justice as Fairness. *Cambridge: MA: Harvard University Press.*

Sandhu R. and Munawer, Q. (1998) How to do discretionary access control using roles, In Proceedings of the *Third ACM Workshop on Role-Based Access Control (RBAC 1998)*.

Sandhu, R. Coyne, E. Feinstein, H. and Youman, C. (1996) Role-Based Access Control Models, *IEEE Computer 29(2).*

Simpson, A. (2008). On the need for user-defined fine-grained access control policies for social networking applications. *Workshop on Security in Opportunistic and social networks.*

Whitworth, B., and deMoor, A. (2003). Legitimate by design: Towards trusted virtual community environments. *Behaviour & Information Technology, vol. 22, no. 1*, 31-51.

Whitworth , B., & Bieber, M. (2002). Legitimate Navigation Links. *ACM Hypertext 2002, Demonstrations and Posters*, (pp. 26-27).

Whitworth, B. (2009). The social requirements of technical systems. In B. Whitworth, & A. De Moor, *Handbook of Research on Socio-Technical Design and Social Networking Systems.* Eds. Hershey, PA: IGI.

Whitworth, B., & Liu, T. (2009). Channel email: Evaluating social communication efficiency. *IEEE Computer*.

Wu, M. Ke, C. and Tzeng, W. (2008) Applying context-aware RBAC to RFID security management for application in retail business, *Proceedings of the IEEE Asia-Pacific Services Computing Conference*.

Access Control Model Instance *

Subject (type : subject, identifier : number, name : string)	*User (subject, number, user)*
Role (subject, number, role)	*Right (type : right, identifier : number, name : string)*
Object (type : object, identifier : number, name : string)	
User (subject U, #1, Alice)	*User (subject U, #2, Bob)*
User (subject U, #3, Carl)	*User (subject U, #4, David)*
User (subject U, #5, Eric)	*User (subject U, #6, Frank)*
User (subject U, #7, Greg)	*User (subject U, #8, Harry)*
User (subject U, #9, Ian)	*Object (object O, #10, o1)*
Object (object O, #11, o2)	*Object (object O, #12, o3)*
Right (right R, #13, View)	*Right (right R, #14, Append)*
Right (right R, #15, Delete)	*Right (right R, #16, Edit)*
UserRoles (subject UR, #17, Family)	*UserRoles (subject UR, #18, Friend)*
UserRoles (subject UR, #19, Colleague)	
GenericRole (subject GR, gr#1, Parent, o1)	*GenericRole (subject GR, gr#2, Child, o1)*
GenericRole (subject GR, gr#3, Parent, o2)	*GenericRole (subject GR, gr#4, Child, o2)*
GenericRole (subject GR, gr#5, Parent, o3)	*GenericRole (subject GR, gr#6, Child, o3)*
ActiveRole (subject AR, ar#1, U : #2 (Bob), R: #19 (Colleague))	*ActiveRole (subject AR, ar#1, U : #3 (Carl), R: #19 (Colleague))*
ActiveRole (subject AR, ar#2, U : #4 (David), R: #17 (Family))	*ActiveRole (subject AR, ar#2, U : #5 (Eric), R: #17 (Family))*
ActiveRole (subject AR, ar#3, U : #6 (Frank), R: #18 (Friend))	*ActiveRole (subject AR, ar#3, U : #7 (Greg), R: #18 (Friend))*
ActiveRole (subject AR, ar#4, U : #4 (David), GR: gr#1 (Parent))	*ActiveRole (subject AR, ar#5, U : #2 (Bob), GR: gr#4 (Child))*
ActiveRole (subject AR, ar#6, U : #1 (Alice), GR: gr#3 (Parent))	*ActiveRole (subject AR, ar#7, U : #1 (Alice), GR: gr#5 (Parent))*
Auth (S : ar#4 (Parent), O : #10(o1), R : #13 (view), ϒ: +)	*subject (U, #4, David)*
Auth (S : ar#4 (Parent), O : #10(o1), R : #15 (delete), ϒ: +)	*subject (U, #4, David)*
Auth (S : ar#3 (Friend), O : #10(o1), R : #16 (edit), ϒ: +)	*subject (U, #6, Frank), subject (U, #7, Greg)*
Auth (S : ar#2 (Family), O : #10(o1), R : #13 (view), ϒ: +)	*subject (U, #4, David), subject (U, #5, Eric)*
Auth (S : ar#1 (Colleague), O : #10(o1), R : #13 (view), ϒ: +)	*subject (U, #2, Bob), subject (U, #3, Carl)*
Auth (S : #8 (Harry), O : #10(o1), R : #13 (view), ϒ: +)	*subject (U, #8, Harry)*
Auth (S : ar#6 (Parent), O : #11(o2), R : #13 (view), ϒ: +)	*subject (U, #1, Alice)*
Auth (S : ar#6 (Parent), O : #11(o2), R : #15 (delete), ϒ: +)	*subject (U, #1, Alice)*
Auth (S : ar#5 (Child), O : #11(o2), R : #13 (view), ϒ: +)	*subject (U, #2, Bob)*
Auth (S : ar#3 (Friend), O : #11(o2), R : #16 (edit), ϒ: +)	*subject (U, #6, Frank), subject (U, #7, Greg)*
Auth (S : ar#2 (Family), O : #11(o2), R : #13 (view), ϒ: +)	*subject (U, #4, David), subject (U, #5, Eric)*
Auth (S : ar#1 (Colleague), O : #11(o2), R : #13 (view), ϒ: +)	*subject (U, #2, Bob), subject (U, #3, Carl)*
Auth (S : #8 (Harry), O : #10(o1), R : #13 (view), ϒ: +)	*subject (U, #8, Harry)*
Auth (S : ar#7 (Parent), O : #12(o3), R : #13 (view), ϒ: +)	*subject (U, #1, Alice)*
Auth (S : ar#7 (Parent), O : #12(o3), R : #15 (delete), ϒ: +)	*subject (U, #1, Alice)*
Auth (S : #3 (Carl), O : #12(o3), R : #13 (view), ϒ: +)	*subject (U, #3, Carl)*
Constraints (C: #c1, R: #14 (Append) ╠ R: #13 (View))	*Constraints (C: #c2, R: #15 (Delete) ╠ R: #13 (View))*
Constraints (C: #c3, R: #16 (Edit) ╠ R: #14 (Append))	*Constraint (C: # c4), GR : gr#1 (parent) ╠ GR: gr#2 (Child)*
Constraint (C: # c4), GR : gr#3 (parent) ╠ GR: gr#4 (Child)	*Constraint (C: # c4), GR : gr#5 (parent) ╠ GR: gr#6 (Child)*

Authorization Set

(ar1, o1, View)	*(UR: #19, O : #10, R: #13, ϒ:+)*	*(ar2, o1, View)*	*(UR: #17, O : #10, R: #13, ϒ:+)*
(ar3, o1, Edit)	*(UR: #18, O : #10, R: #16, ϒ:+)*	*(ar4, o1, Delete)*	*(GR: gr#1, O : #10, R: #15, ϒ:+)*
(Bob, o1, View)	*(U: #2, O : #10, R: #13, ϒ:+)*	*(Carl, o1, View)*	*(U: #3, O : #10, R: #13, ϒ:+)*
(David, o1, View)	*(U: #4, O : #10, R: #13, ϒ:+)*	*(Eric, o1, View)*	*(U: #5, O : #10, R: #13, ϒ:+)*
(Frank, o1, View)	*(U: #6, O : #10, R: #13, ϒ:+)*	*(Greg, o1, View)*	*(U: #7, O : #10, R: #13, ϒ:+)*
(Frank, o1, Append)	*(U: #6, O : #10, R: #14, ϒ:+)*	*(Greg, o1, Append)*	*(U: #7, O : #10, R: #14, ϒ:+)*
(Frank, o1, Edit)	*(U: #6, O : #10, R: #16, ϒ:+)*	*(Greg, o1, Edit)*	*(U: #7, O : #10, R: #16, ϒ:+)*
(David, o1, Delete)	*(U: #4, O : #10, R: #15, ϒ:+)*	*(Harry, o1, View)*	*(U: #8, O : #10, R: #13, ϒ:+)*
(ar1, o2, View)	*(UR: #19, O : #11, R: #13, ϒ:+)*	*(ar2, o2, View)*	*(UR: #17, O : #11, R: #13, ϒ:+)*
(ar3, o2, Edit)	*(UR: #18, O : #11, R: #16, ϒ:+)*	*(ar5, o2, View)*	*(GR: gr#4, O : #11, R: #13, ϒ:+)*
(ar6, o2, Delete)	*(GR: gr#3, O : #11, R: #15, ϒ:+)*		
(Bob, o2, View)	*(U: #2, O : #11, R: #13, ϒ:+)*	*(Carl, o2, View)*	*(U: #3, O : #11, R: #13, ϒ:+)*
(David, o2, View)	*(U: #4, O : #11, R: #13, ϒ:+)*	*(Eric, o2, View)*	*(U: #5, O : #11, R: #13, ϒ:+)*
(Frank, o2, View)	*(U: #6, O : #11, R: #13, ϒ:+)*	*(Greg, o2, View)*	*(U: #7, O : #11, R: #13, ϒ:+)*
(Frank, o2, Append)	*(U: #6, O : #11, R: #14, ϒ:+)*	*(Greg, o2, Append)*	*(U: #7, O : #11, R: #14, ϒ:+)*
(Frank, o2, Edit)	*(U: #6, O : #11, R: #16, ϒ:+)*	*(Greg, o2, Edit)*	*(U: #7, O : #11, R: #16, ϒ:+)*
(ar7, o3, Delete)	*(GR: gr#5, O : #12, R: #15, ϒ:+)*	*(Alice, o3, View)*	*(U: #1, O : #12, R: #13, ϒ:+)*
(Alice, o3, Delete)	*(U: #1, O : #12, R: #15, ϒ:+)*	*(Carl, o3, View)*	*(U: #3, O : #12, R: #13, ϒ:+)*

**Owner has all the rights over the object so owner role is not considered*

Figure 2: Access control Model Instance and its authorization set