Dynamic Rights Reallocation In Social Networks[1]

Adnan Ahmad[1], Brian Whitworth[1] and Lech Janczewski[2]

[1] Massey University, Auckland, New Zealand
2 The University of Auckland

[1] Published as A. Ahmad, B. Whitworth and L. Janczewski, *Dynamic Rights Reallocation in Social Networks*, Human Factors in Information Security and Assurance (HAISA), in International Information Security and Privacy Conference (IFIP SEC'12), Heraklion, Crete, Greece, June 4-6, 2012

Abstract. Access control, as part of every software system, has evolved as computing has evolved. Its original aim was to limit unauthorized access for centralized systems, but the rise of online social networks like Facebook has changed that. Now each person wants to control who sees photos or makes comments on their local wall by making and unmaking friends, i.e. dynamic, distributed rights control. Social networks already have access control, but there is currently no agreed logical model for their rights, no consistent scheme for allocating and re-allocating permissions to create, edit, delete and view social objects and entities. A socio-technical approach based on social and technical requirements can give the basics of a model. Various rights re-allocations like transfer, delegate, divide and multiply are explored. It suggests a theoretical base for access control beyond its security parent.

Keywords
Social Networks, Rights analysis, Rights reallocation.

1. Introduction

The need for access control arose with multi-user computing, as users sharing the same system came into conflict [1]. As computing evolved, access control logic developed to offer *local domain* access control for distributed systems, and *user roles* for systems with many users. With variations, the traditional access control approach has worked for military and commercial applications, organizational structures, contextual decisions, distributed applications, medical data, peer-to-peer networks and the grid environment [20-23].

The last decade has seen extreme multi-user systems emerge - *social networks* (SNs) where millions of users share billions of resources and grant each other access rights [2]. As access control now depends on the number of interactions, its complexity increases geometrically with size, not linearly. Mapping millions of subjects directly to billions of resources is unwise, as each account adds hundreds or thousands of photos and comments a year. 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.

As social networks are here to stay, and growing in number and size, a logical model of distributed rights allocation is needed. The aim is to identify software *patterns* that embody social principles as well as technical principles like efficiency [2]. The result would be a consistent scheme to allocate and re-allocate distributed rights in a socially acceptable way. The rest of the paper is organized as follows: Section 2 reviews previous work, Section 3 gives the specifications, Section 4 presents the model, Section 5 analyses it, and Section 6 discusses it.

2. Review

A semi-decentralized access control model is presented in [3] where users are categorized in terms of relationship depth and trust level. Likewise dRBAC manages trust in coalition environments by decentralized access control [4]. Additionally, some other access control solutions use trust [5], reputation [6] and relationships [7] to manage access rights between users. However, there is no access control model for SN that supports rights reallocation.

On the other hand, there exist some models of delegation for traditional access control systems, but other types of reallocation like rights sharing, division and transfer have hardly got any attention. Existing delegation models can be categorized into machine to machine [10 del], user to machine [11 del], and user to user [12 del]. These models deal with the mutual delegation of objects [10 del] – one object acting on other's behalf, user to object [11 del] – objects acting on user behalf, and role delegation [12del] – user assigning roles to other users. Traditional models cannot be mapped on current SN due to the following reasons:

Traditional solutions see SN access requirements through a security lens, so do not give local control over user contributions like family photos, and so struggle with privacy demands. Central access control gives each user the same policy, so variants must be requested from a central authority who sets system wide roles. The user has no local control over their resources, as friends can't be specified by generic roles.

The delegation proposed in literature works on system wide entities, but SN introduced local autonomous domains. So domain based delegation is required rather than role based. Furthermore, current models provide single multilevel delegation which is most suitable for roles, but SN require multiple single level delegations to maintain accountability and delegate subparts of domain.

Current access control models for SN do not specify the dynamic allocation of distributed rights found in social networks [12, 13], where *everyone* can give rights away. In dynamic, distributed control, each person can fully administer their own domain. Previous work to develop a logical rights framework does not cover rights reallocation [2, 14, 15], but in social networks friends are regularly made and unmade, i.e. managing rights transfer is a critical success criterion.

3. Specifications

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 involves technical and social requirements, to model not just what *can* be done but what *should* be done.

3.1 Overview

An information system has *entities* and *operations*, where:

1) ***Entity***. Passive stored information, i.e. data.
    a) *Social entity.* Represents an accountable party.
        i) *Persona*. Represents an external person or group.
        ii) *Agent*. Acts on behalf of another social entity.
    b) *Object.* Can convey information to a social entity.
        i) *Item.* A simple object with no dependencies, e.g. a bulletin board post.
        ii) *Space.* A complex object with dependents, e.g. a bulletin board thread or Facebook wall.
    c) *Right.* A system permission for a social entity to operate on an information entity.
        i) *Simple rights*. Apply to entities.
        ii) *Meta-rights*. Rights to rights, e.g. delegate.
2) ***Operation***. A program is active information. Subject-object acts can be classified as
    a) *Passive* acts don't change a target, e.g. view, enter.
    b) *Use* acts change a target, e.g. edit, create.
    c) *Communication* acts target a social entity, e.g. send.
    d) *Social* acts target rights or roles, e.g. delegate.

3.2 Reallocating Rights

The ability to reallocate social rights is the key to meet social requirements. It allows socio-technical systems to evolve from an initial state of one administrator with all rights to a community with delegated and shared rights. Allocation can change the actors in a right or role as follows:

1) *Transfer.* Allocate use and meta-rights and is irrevocable.
2) *Delegate.* Allocate use rights only and is revocable.
3) *Divide.* Allocate rights jointly to an actor set.
4) *Multiply.* Allocate rights severally to an actor set.

If a right is owned *jointly,* all must agree to allow the act, while if it is owned *severally*, any party alone can activate it. The above can act in combination, e.g. to transfer joint ownership. Ancestor and offspring roles are unaffected by owner reallocations. Table 1 shows the details, as follows:

1. *Transfer*. Transfer gives all entity rights, including meta-rights [16]. Rights are irrevocably given to the new owner, e.g. after selling a house, the old owner has no rights to it.

2. *Delegate.* Delegate gives use rights but not meta-rights, so can be taken back, e.g. a system administrator who delegates rights can take back the top system priority [17].
3. *Divide.* Those who divide ownership jointly own an entity, e.g. a couple who jointly own a house must both agree to sell it. In joint ownership, any party can stop an act.
4. *Multiply.* In multiply the entire right is given complete, so any party can act alone as if they owned it exclusively, e.g. a couple's bank account where both can withdraw all the money.

Table 1. Allocating use and meta rights

	Allocated by			
	Meta rights	Use rights	Meta rights	Use rights
Transfer			√	√
Delegate	√			√
Divide use	√	½√		½√
Divide all	½√	½√	½√	½√
Multiply use	√	√		√
Multiply all	√	√	√	√

For example, a many author paper submitted online can let *one* author alone edit it (transfer), let *one* author edit as allowed by the primary author (delegate), let edits proceed only if confirmed by *all* authors, or let *any* author do any edit. The model covers the social options.

If a delegatee gets no meta-rights, they can't pass rights on, e.g. renting an apartment gives no right to sub-let[2]. Similarly, lending a book to another doesn't give them the right to on-lend it, though as with all social requirements, it happens. Yet being consistent maintains accountability, e.g. if one loans a book to a person who loans it to another person, who then loses it, who is accountable to the original owner? This gives the operational principle:

P1. Delegating doesn't give the right to delegate.

A right reallocation is *revocable* if the initiating party keeps the meta-rights, so delegation is revocable but not transferable. Dividing use rights is revocable but dividing all rights is not, as reverting would require joint agreement. Multiplying

2

By this social logic, lessees can't sub-let, i.e. delegate tenancy rights on to others.

use rights is revocable but multiplying meta-rights is a dictator's dream case as anyone can allocate all rights to anyone. It is likely unstable.

To allocate a right to an existing object makes one accountable for it, so by fairness requires consent, e.g. one doesn't add a paper co-author without their agreement. The principle is:

P2. Allocating use rights to existing objects requires consent.

One can't make someone the owner of something unless they agree. The ACS would have to put a question like: "*Martin wants to transfer edit rights to object to you, do you agree?*" In contrast, rights with no accountability for existing objects can be allocated without permission, as the other can use them if they wish, e.g. view, enter and create. The principle is:

P3. Rights that imply no existing objects can be allocated freely.

So space owners can delegate entry, view and create rights without inconsistency. These are social requirements not technical necessities. As technical requirements express technical good practice, so social requirements express social good practice. For best effect, they should be applied consistently.

3.3 Social networks

The model both clarifies how social networks operate and suggests alternatives, e.g. social networks send messages like:

"*X wants to be friends with you*"

In this model, it is a social trade: X will add you to their friend role if you add them to yours. It can be handled as a two-step social transaction, but the steps need not be linked. A tit-for-tat is assumed, but one can *befriend* another, i.e. add them to a friend role, without their permission (P3). One could make another a friend, with view rights, whether they return the favor or not. In future, one could receive messages like:

"*X has made you a friend* "

This is an offer to *be a friend*, not a request to *be my friend*. As one can love another who doesn't return the favor, so friendship needn't be mutual. Systems that axiomatize friendship mutuality limit it, as socially friendship is given not taken.

Equally, to make friends of my friends also my friends is to contradict P1 that giving a right doesn't give a meta-right. As liking someone doesn't guarantee that one will like their friends, making another a friend doesn't grant them access to my friend list. This is a technical option that has no social basis.

4. The Formal Model

An access control matrix can be expressed using some function *Grant-Right (A, O, R)* which holds whenever the access control matrix gives right *R* to actor *A* over object *O*. So, the function of the form

Grant-Right (Alice, abc.txt, View)

states that Alice can view the abc.txt file. This kind of simple function can be used to assign various rights to roles instead of individual actors, e.g. to allow the destroy right to ancestor role over file abc.txt, the function will look like

Grant-Right (Ancestor, abc.txt, Destroy)

Also, some rights are more powerful than others and contain others as their subset, like delete right includes view right in it or allocating edit right gives append right to the same actor. This type of rules imply

Grant-Right (A, O, Edit) ╟ *Grant-Right (A, O, Read)*

which makes the edit right stronger than the read right, but nothing else. Apart from the basic *Grant-Right* function which is a nice way to represent the rights stored in the access control matrix, access control logics include formulas of the form *A says* Ω, where *A* is an actor and Ω is a formula [9]. The formula represents that actor *A* makes statement Ω, which can be a request, assignment of some rights to some other actor or role, or as a part of security policy. For example, the administrator *admin* of a domain may certify that the creator of an object *O* is its owner; this assertion may be represented as

Admin says Owner (Object)

However, the minimum necessary condition of using the *say* function is that the authorization actor must hold the right that is given away to the other actor by him. In the above statement, it is implicit that *admin* is the owner (or authorized actor) of the system and he is authorizing another role *Owner*.

Now the framework in formal access control logic will be formulated for the above mentioned rights using the following scenario: A(lice) assigns G(reg) and F(rank) as friends, D(avid) and E(ric) as family, and B(ob) and C(arl) as colleagues. Also, D(avid) sees H(arry) as a friend and B(ob) sees I(an) as a friend.

Suppose that Alice creates one file *o1* in David space *CO* and one file *o2* in her own space *CO*. 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 Table 2.

Ownership can be modeled in the natural way in the presented framework as it does not distinguish the administrative authority. So, the condition formula *A says Φ* can be used in open or closed environment in the same manner, where both *A* and Φ are arbitrary formulas. The model supports *owner says* Φ in the same manner as *admin says* Φ, if the initial system instance supports ownership.

Joint ownership can be modeled in the same fashion as the single ownership using the extension of the same formula owner formula in the form $A \wedge B$ *says* Φ to mean that principal A and B jointly says Φ. This would require the consent of both A and B to execute a function Φ, where Φ can be any arbitrary operation legal in the settings of access control model instance.

Sharing can be used by extending the same ownership formula in the form $A \vee B$ *says* Φ to mean that principal A or B *says* Φ. In sharing, Φ needs to be explicitly defined for the object as sharing allocates some of the rights to the secondary owner but not all, but the actors can execute the shared operation alone on their own behalf.

Transfer is an action upon a right, to change its actor property. Giving a right simply changes the actor for all rights. Delegating a right does it for all but the entity meta-right. Dividing a right replaces the actor by an AND set. Multiplying a right replaces the actor by an OR set.

Delegation is also executed in multiple operations as the addition of all the rights to the delegatee including the *says* operation, but the revocation of the delegation method remains with the delegator. The second step is the removal of edit and some other rights from the delegator access matrix so the delegatee is the only responsible actor over the state of the object.

6. Conclusions

This paper suggests how to allocate and reallocate access control rights to satisfy social requirements like creator ownership. The formal semantics and syntax of an access model were given, to show it is efficient, consistent and reachable. Examples given from Facebook, YouTube and others suggest that this is already occurring.

This highlights the fact that while access control began in the shadow of security, in socio-technology it will become a separate discipline. While security needs secrecy for obvious reasons, access control should be public, so people can see what the rules are, In physical society laws are *visible* to all not hidden away, as signposting social requirements is better than letting people make social errors to embarrass or punish them. The social requirement of transparency demands access control rules to made public, so actors can anticipate and avoid social errors, to reduce governance corruption and to increase social trust, as people recognize the permissions of others [18].

The next phase of this project is to develop a distributed, dynamic access control plug-in for a NSF granted open knowledge exchange (OKE) system and evaluate it with respect to both social criteria like fairness and technical criteria like storage efficiency [19]. This "rights module" will also give human readable reports to tell actors of granted rights, i.e. be transparent. The goal is that social rights are not only applied but also seen to be applied, as this is critical for trust and synergy.

Online communities today can't survive without participation, so access control is increasingly about access rather than control, i.e. about letting people in rather than keeping them out. This model follows the socio-technical paradigm: to first define the social requirements, then design a technical solution to meet them. It avoids at source the social errors of technical design, like spam, and the chance of online social success. The evolution of access control beyond security will open up new research dimensions.

Acknowledgment

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”.

References

[1] Alan H. Karp et al. From ABAC to ZBAC: The Evolution of access control models, 2009

[2] A. Ahmad and B. Whitworth, "Distributed Access Control for Social Networks", In proceedings of International conference of information assurance and security IAS, December 2011.

[3] B. Carminati, E. Ferrari, and A. Perego, “Enforcing access control in web-based social networks” ACM Transactions on Information & System Security, 2008.

[4] E. Freudenthal, T. Pesin, L. Port, E. Keenan, V. Karamcheti, “dRBAC: Distributed role based access control for dynamic coalition environments” In ICDCS '02: Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02), 2002.

[5] Ali, B., Villegas, W., and Maheswaran, M. ,” A trust based approach for protecting user data in social networks”. In 2007 Conference of the Center for Advanced Studies on Collaborative research (CASCON’07), pages 288–293, 2007.

[6] Carminati, B., Ferrari, E., and Perego, A. ,” Rule-based access control for social networks”. In On the Move to Meaningful Internet Systems 2006: OTM Workshops 2006.

[7] Tapiador, A., Carrera, D. and Salvachúa, J., "Tie-RBAC: an application of RBAC to Social Networks". Web 2.0 Security and Privacy, Oakland, California, 2011.

[8] Martín Abadi: Logic in Access Control (Tutorial Notes). FOSAD 2009: 145-165.

[9] Valerio Genovese, Laura Giordano, Valentina Gliozzi, Gian Luca Pozzato. A constructive conditional logic for access control: a preliminary report. ECAI'2010. pp.1073~1074.

[10] Deepak Garg, Martín Abadi: A Modal Deconstruction of Access Control Logics. FoSSaCS 2008: 216-230

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

[12] B. Carminati, E. Ferrari, and A. Perego, "Security and privacy in social networks," In Encyclopedia of Information Science and Technology, 2nd Edition, volume VII, pages 3369–3376. IGI Publishing, Sept. 2008.

[13] A. Simpson, "On the need for user-defined fine-grained access control policies for social networking applications," In SOSOC '08: Proc. of the Workshop on Security in Opportunistic and social networks, New York, USA, 2008.

[14] Whitworth, B., Aldo de Moor, A. and Liu, T., 2006, Towards a Theory of Online Social Rights, in R. Meersman, Z. Tari, P. Herrero et al. (Eds.): OTM Workshops 2006, LNCS 4277, pp. 247 – 256, Springer-Verlag Berlin Heidelberg.

[15] Whitworth, B., & deMoor, A. (2003). Legitimate by design: Towards trusted virtual community environments. Behaviour & Information Technology, 22:1, p31-51. Journal

[16] Gaaloul, K., Zahoor, E., Charoy, F., & Godart, C. (2010). Dynamic Authorisation Policies for Event-based Task Delegation.

[17] Gaaloul, K., Schaad, A., & Flegel, U.. A secure task delegation model for workflows. In Proceedings of The Second International Conference on Emerging Security Information, Systems and Technologies, 2008.

[18] J. Kooiman, M. Bavinck, R. Chuenpagdee, R. Mahon, R. Pullin, "Interactive governance and governability: an introduction," The Journal of Transdisciplinary Environmental Studies, 7(1), 2008.

[19] National Science Foundation (NSF), award number 0968445. "OKES: An open knowledge exchange system to promote meta-disciplinary collaboration based on socio-technical principles", 2010-2011.

[20] Lampson, B. W., "Dynamic Protection Structures," AFIPS Conference Proceedings, 35, 1969, pp. 27–38.

[21] TCSEC, Trusted Computer Security Evaluation Criteria (TCSEC), DOD 5200.28-STD. Department of Defense, 1985.

[22] Clark, D. D., and D. R. Wilson, “A Comparison of Commercial and Military Computer Security Policies,” IEEE Symposium of Security and Privacy, 1987, pp. 184–194.

[23] Ferraiolo, D., and D. R. Kuhn, “Role-Based Access Control,” in Proceedings of the NIST-NSA National (USA) Computer Security Conference, 1992, pp. 554–563.

Figure 2: Access control Model Instance and its authorization set

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*