Intentional Access Management: Making Access Control Usable for End-Users

Xiang Cao
University of British Columbia,
Dept. of ECE, 2332 Main Mall, Vancouver BC, Canada
1-604-827-5909
xiangc@ece.ubc.ca

Lee Iverson
University of British Columbia,
Dept. of ECE, 2332 Main Mall, Vancouver BC, Canada
1-604-822-3381
leei@ece.ubc.ca

ABSTRACT
The usability of access control mechanisms in modern distributed systems has been widely criticized but little studied. In this paper, we carefully examine one such widely deployed access control mechanism, the one embedded in the WebDAV standard, from the point-of-view of an end-user trying to decide how to grant or deny access to some resource to a third party. This analysis points to problems with the *conceptual usability* of the system. Significant effort is required on the part of the user to determine how to implement the desired access rules; the user, however, has low interest and expertise in this task, given that such access management actions are almost always secondary to the collaborative task at hand. The analysis does however indicate a possible solution: to recast the access control puzzle as a decision support problem in which user intentions (i.e. the descriptions of desired system outputs) are interpreted by an access mediator that either automatically or semi-automatically decides how to achieve the designated goals and provides enough feedback to the user. We call such systems *intentional access management* (IAM) systems and describe them in both specific and general terms. To demonstrate the feasibility and usability of the proposed IAM models, we develop an intentional access management prototype for WebDAV. The results of a user study conducted on the system show its superior usability compared to traditional access management tools like the access control list editor.

Categories and Subject Descriptors
D.4.6 [**Operating Systems**]: Security and Protection – *access controls.* H.1.2 [**Models and Principles**]: User/Machine Systems – *human factors, human information processing.*

General Terms
Design, Security, Human Factors.

Keywords
Access control, usability, intentional access management, WebDAV.

1. INTRODUCTION
Technological advances in computers and networks, especially the proliferation of the Internet, have made it easier for novice or average end-users to share their information and collaborate with others. However, with this unprecedented ease of sharing information, it becomes increasingly challenging for users to protect their shared information against unauthorized use. Compared with professional security administrators, end-users usually have limited technical capacity and less security knowledge, and so find it more challenging to properly use and/or maintain the security mechanism provided to control access to their resources [1] [2] [15].

To end-users, security is usually a secondary goal [15]. People just want security in place to protect them while they are achieving their primary goal (e.g., browsing web pages, co-authoring papers, etc.). Sometimes security is a "necessary evil" (i.e., something a user is required to do but is not interested in doing). In addition, security tasks may not be everyday tasks for end-users. They may need to be done every few weeks or months. If the operations of managing security are arcane, end-users will have difficulty in remembering them for the next time [11]. Further, making security usable is actually an enabling task which can encourage end-users to share information [14].

We are primarily interested in security for information sharing in modern cyberspace, where access control remains a major challenge. Therefore, in this paper we limit the scope of our discussion to making access control more usable for end-users, although some of the results may also apply to other security usability problems. In particular, to support information sharing, we choose shared file-systems as the underlying collaborative infrastructure. In this context, there is a need for fine-grained, user-centered, dynamic control of sharing to match trust and ad-hoc collaboration. The solution we are pursuing is the end-user management of resource sharing with minimal changes to backend infrastructure. Making such management usable enough to be effective for those non-expert end-users then becomes our main task.

The traditional access controls include mandatory access control (MAC), discretionary access control (DAC), and role-based access control (RBAC) [13]. To implement the access controls, access control lists (ACLs), capability lists, or policy-based mechanisms are often adopted. Since our research interest is information sharing that can be controlled by end-users, in this

paper we are looking at discretionary access control with a potential variety of enforcement mechanisms.

In this paper, we use the term, "*access management*," to refer to the process of formulating the access control rules (rule sets, group memberships, attributes, etc.) for an access control system to achieve a desired set of access privileges for a user or users. This access management is performed by a user interacting with an access control system. In a typical access control system, users interact with the system through a graphic user interface (GUI), which allows them to manipulate the internal access control mechanism directly or indirectly. Most of the management burden, however, such as formulating the rules based on his or her security goals is carried by the end-user. Given end-users low interest in security, security systems that place less of a burden on them for access management should be inherently more usable. In this paper we will demonstrate that the management workload laid on the user is usually too heavy, even for a simple access management task, if no appropriate decision support is provided to the user. To address this problem, some novel access management models, called *intentional access management models*, which isolate end-users from access control mechanisms, are then proposed.

To demonstrate the usability problems in current access control systems and the feasibility of our proposed solution, we have chosen an existing access control mechanism, the WebDAV access control list standard [4], as the starting point for usability improvement. In particular, the WebDAV (Web-based Distributed Authoring and Versioning) [7] system is designed to create an infrastructure for distributed file systems built on top of the now-ubiquitous foundation of the World-Wide-Web, the HTTP protocol. With this system, it becomes possible to think of an entirely user-controlled distributed information sharing environment with enormous flexibility and potential [6].

Since targeting such a widely deployed Internet-scale specification makes potential results more applicable to real-world information-sharing settings, and WebDAV access control semantics are modeled after Windows NTFS access control semantics (with some simplifications), we have chosen WebDAV access control as the basis for our research on usable security. The results should also have implications for improving the usability of similar access control systems, such as Windows NTFS access control.

We make two simplifying assumptions. First we assume the access control is supported by a good and usable authentication scheme and this paper will not discuss authentication. Second, we assume that the security system is in a multi-user environment and access control of a resource is managed by a combination of a central administrator and the end-users who have the appropriate right to perform the management. Although some of our analysis can be applied to systems that are entirely centrally administered, we will not concern ourselves with these in this paper. Instead we focus on end-user control since such systems are becoming more and more prevalent and necessary.

The rest of the paper is organized as follows. Section 2 picks WebDAV access control as an example to present our detailed analysis of the access management workload of the user. To remove some of the load from the user, Section 3 then introduces a set of design principles and three levels of models for usable access management which satisfy these principles. The design and implementation of an access management system for WebDAV that embodies the proposed models are described in Section 4. Section 5 presents a user study on managing access to a WebDAV repository, demonstrating the usability improvement of the new design for end-users. Some related work is discussed in Section 6. Finally Section 7 summarizes the paper and suggests further work.

2. INTENTIONAL ANALYSIS OF WEBDAV ACCESS CONTROL

The administrator/end-user who wants to control the access to his/her resources first only has a goal or intention such as who can perform what operation on what resource in mind. However, intention is not implementation. Only by a sequence of processes of checking and analyzing the state of the concerned system will he/she reach a decision on how to resolve this intention (e.g., designing the ACLs). Below, we will take WebDAV access control which employs a simple ACL model as a case study to demonstrate that there is a large gap between the intention and the actual implementation of control for the user. In this paper either the administrator or the end-user who has the right to manage the access control is called "user" indistinguishably.

2.1 WebDAV Access Control

The access control list model for WebDAV resources was very consciously derived from existing file system practice, with slight adaptations to handle resource properties. The resources are organized in a filesystem-like hierarchy. Access is divided into several well-defined privileges (e.g., read, write). Privileges may be containers of other privileges, in which case they are termed "aggregate privileges". Each resource has one ACL. The ACL contains a set of "access control entries/elements" (ACEs), where each ACE specifies a principal and a set of privileges that are either *granted* or *denied* to that principal on this resource. An ACE can be inherited from the ACL of another resource. A principal may also be a group, where a group is a collection of other principals, called the members of the group. The groupings of principals, privileges, and resources by hierarchy, complicate the access control list model.

Usually WebDAV access control systems provide a user interface (UI) that allows users to manipulate the ACLs, like the Windows systems using NTFS do. To manage the access control by manipulating the ACLs, the user has to learn how an ACL is evaluated to determine whether or not access will be granted for a WebDAV request. ACEs are maintained in a particular order, and are evaluated until all of the permissions required by the current request have been granted, at which point the ACL evaluation is terminated and access is granted. If, during ACL evaluation, a <deny> ACE (matching the current requesting principal) is encountered for a privilege which has not yet been granted, the ACL evaluation is terminated and access is denied. Failure to have all required privileges granted results in access being denied.

The evaluation procedure described above leads to the distinction between the *stated privileges*, the explicit privileges granted or denied to the user in the ACEs, and *effective privileges*, the actual privileges that a user will have according to the combination of the involved ACEs in an ACL. A similar distinction exists in the Windows NTFS ACL model. The confusion between stated and

effective privileges is one of the main sources of error for end-users in manipulating the ACL, which was observed in our user study. Maxion and Reeder [11] have proposed a new interface called Salmon for setting NTFS file permissions, which mitigates such human errors by presenting needed information including the effective permissions in the GUI to the user. However, through the following analysis, we will show that only providing such information is not sufficient for end-users.

2.2 Intentional Analysis

So, how can we assess or improve the usability of this ACL model? Although a GUI ACL editor like the one suggested in [18] can facilitate manipulating and understanding of the ACLs, and the WebDAV ACL model is relatively simple, we suggest that before even considering user interface issues we should determine whether too much may be asked of the user in terms of sheer involvement with the process. As stated above, we contend that the end-user is primarily interested in the output of the ACL system (e.g., either a success or failure of an access request), and not in the means by which this output is achieved. Since the conceptual complexity of a task shouldn't exceed the user's commitment to that task, it becomes necessary to characterize the complexity and assess how well it matches the user's expectation. In this paper, we suggest approaching it from an algorithmic point of view, since it is easy, straightforward and sufficient for this kind of tasks. Developing and analyzing the algorithm will reveal the algorithmic relationship between the goal and configuration changes. It will help understand where the complexity in the system lies, characterize and reveal representations/visualization of state needed to meet a user's goal.

To assess this complexity and eventually try to match it with the user's commitment, we will start our analysis from the goal states and go backwards to try to determine the minimal changes needed to achieve the intention. In this paper, we will examine two simple goal states that the user is likely to be interested in achieving:

G1: Principal X must have privilege Y on object Z.
G2: Principal X must not have privilege Y on object Z.

The principal here may be a set of principals. We focus on these two intentions since they represent the basic effects all access control systems should produce through the implementation of access control. Also all the goal statements discussed in the user study for Salmon [11] can be generalized into these two intentions, if the privileges in user intentions match the privileges used in the system. We describe these goal states as *primary user intentions* and analyze the steps necessary to determine what ACL changes must be made to the current system state to ensure that the output of the ACL system will produce these results. For obvious reasons, we call this style of analysis "*intentional analysis*". Note that this is not an analysis of the intentions that users will have, but of the process that has to be taken based on the intentions.

The task at hand for the user is then to (1) assess the current state of the system, (2) decide whether or not the goal is already fulfilled, and (3) develop a strategy to decide how to minimally achieve the goal state given the current system state. Since the WebDAV ACL system is relatively simple, it is likely that we can develop an algorithm to determine this minimal change set. The complexity of this algorithm is then a reasonable measure of the complexity of fulfilling this intention. Therefore in this paper we also call the analysis based on the algorithm "algorithmic analysis" and the corresponding task complexity "algorithmic complexity".

2.2.1 ACLs in Collaborative Environment

Consider that the state of ACL system is the result of a number of people with access to shared file-systems collaborating on the current configuration. If we assume that each of these users is pursuing a similar goal-oriented approach to making ACL changes, then the current state should be considered as the result of a multi-agent collaboration. Therefore, before proceeding to the full analysis, let us examine the first stage of the reasoning necessary to deal with the intention *G1*: "Principal X must have privilege Y on object Z." At first glance, it may appear easy to implement – just adding an ACE that grants the particular principal X the privilege Y to the top of the ACL of object Z. Since the order of ACEs determines precedence, this addition will automatically override any <deny> ACEs that might appear later in the ACL. If we suppose however, that the <deny> ACE was added by another user through his or her own intention then we have a conflict between these two intentions that both users should be informed of. Detecting and resolving such conflicts may be important in a collaborative environment. Therefore, rather than simply adding the grant at the top, we should examine all ACEs until we find one that specifically grants or denies the access rights we intend. Some ACL systems, such as the Windows NTFS ACL system, specify that <deny> ACEs always take precedence over <grant> ACEs. In this case, adding a <grant> ACE will never fulfill the intention if conflicting <deny> ACEs exist in the ACL.

In addition, simply adding an ACE to the top of an ACL without examining the existing ACEs may introduce redundancy to the ACL, which may then result in undesired consequences in the long term. For instance, suppose that the existing ACEs have determined that principal X already has privilege Y or some sub-privileges contained in privilege Y, and the user simply adds a <grant> ACE at the top to fulfill the intention *G1*. If later on the user wants to achieve the reversed intention, *G2*, it is possible that he/she may just undo the previous action by removing that <grant> ACE from the ACL. This action will result in an incorrect or incomplete accomplishment of *G2,* since the pre-existing ACEs granting partial rights to *G1* are still in the ACL. Exactly such mistakes were observed in the user study presented in Section 5.

Of course, even if the user does not have direct modification rights to the ACL of object Z, it may be possible to grant the access requested. Due to the indirection implied by group membership, it may be possible to grant the privilege Y by modifying the membership of an existing group that has this privilege. Of course, this indirect grant will have side effects (e.g., there may be other objects that the group has rights to and the principal X does not) and the user should at least be notified of these. In addition, if ACL inheritance is enabled in the system, it may be possible to grant the privilege Y by modifying the inherited ACL, if permitted. This may also cause side effects. Let us consider an example. If user A has the privilege to read a document object Z, and he wants principal X, who does not have the read privilege, to read this document. Unfortunately he finds he does not have the privilege to modify the ACL of Z. If he does

not know the indirect means of granting access described above, he may just send the document Z to principal X as an email attachment. From this point on, the access control system is essentially irrelevant (at least as far as read access is concerned), since a copy of the file is now "in the wild." As an alternative to copying then, side effects may be acceptable.

So, even without considering ACL inheritance (which we will ignore for the purposes of the flow-chart analysis below), we have conflicts that should at least be noticed, and potential side effects from trying to resolve an intention. Clearly resolving even these simple intentions is not a trivial problem.

2.2.2 Algorithmic Analysis

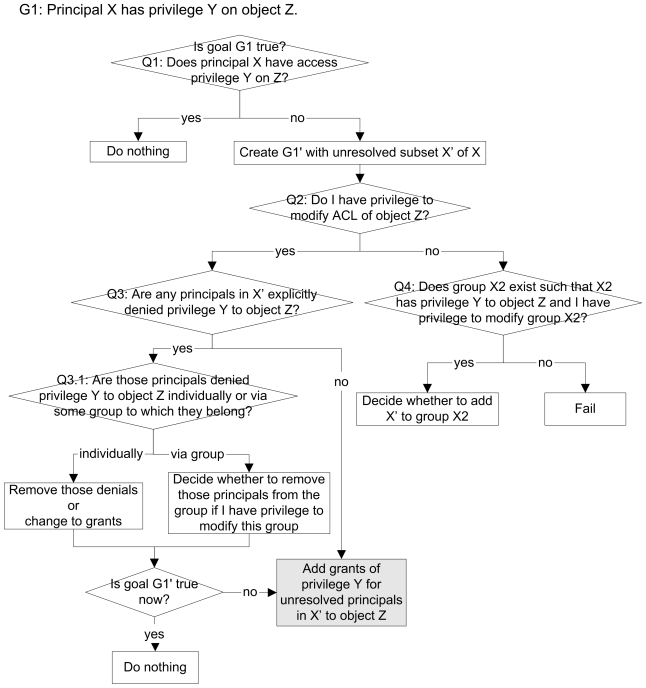

Figure 1. A possible decision process for determining how to implement G1: *principal X must have privilege Y on object Z*

Figure 1 shows a decision process or algorithm that will allow the user to resolve the intention G1 or be given enough information to know why he or she was unable to do so. The process begins with a query Q1 ("Does principal X have access privilege Y on Z?") to check if the goal has already been achieved. If it has not been fully achieved, the user can select all principals X' in X who don't have privilege Y on Z for further processing. The user then needs to check if he has the privilege to modify the ACL of object Z. If he does, then he examines that ACL in the left branch. In this branch, if any of the principals in X' are explicitly denied privilege Y to object Z, that means there are conflicts between the current access control intent and whichever previous intents that caused the denial to be added to the ACL. We will consider the handling of such conflicts in Section 2.2.4. Even if instead the user cannot modify the ACL, the process will not end. As shown in the right branch, the user can check if some group X2 exists such that X2 has privilege Y to object Z and that he has the privilege to modify X2. If the answer is "yes", the user has to make a decision on whether to add X' to this group, because this action may cause side effects: as a member of X2, the principals in X' may have all the privileges group X2 has and lose all the privileges group X2 is denied. These side effects are produced not only on object Z but also on other resources in the system.

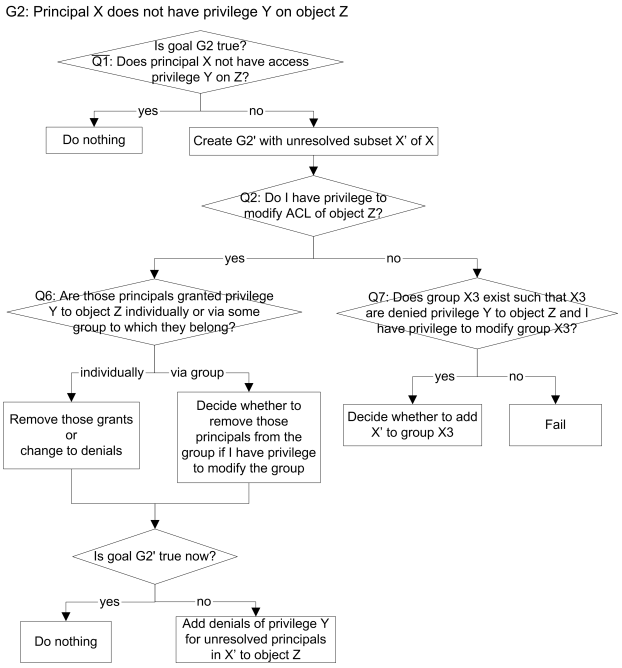

Figure 2. A possible decision process for determining how to implement intention G2: *principal X must not have privilege Y on object Z*

The flowchart in Figure 1 does not have any loops, but it does have six branch points. Combining it with the flowchart in Figure 2 (and Figure 3 below) we can enumerate all of the branch points:

- Q1: Does principal X have privilege Y on Z?
- Q2: Do I have the right to modify the ACL of object Z?
- Q3: Are any of the principals in X' explicitly denied privilege Y to object Z?
- Q3.1: Are those principals denied privilege Y to object Z individually or via some group to which they belong?
- Q4: Does group X2 exist such that X2 has privilege Y to object Z and I have the privilege to modify group X2?
- Q5: Do I have the privilege to create a new group?
- Q6: Are those principals granted privilege Y to object Z individually or via some group to which they belong?
- Q7: Does group X3 exist such that X3 is denied privilege Y to object Z and I have the privilege to modify group X3?

Note that these decision points all involve queries of the system state. If the user does not deeply understand the ACL evaluation model, there is no reason to suppose that he or she would be able to easily resolve these questions. Frankly, it is unlikely that most users would even know to ask many of them.

For example, consider the query Q1 "Does principal X have privilege Y on Z?" Answering this query involves evaluating the effective privileges for principal X. As described above, a principal's effective privileges are computed according to a formula that sums the principal's explicit privileges granted by some ACEs (directly or indirectly via group) and have not been denied by preceding ACEs in the ACL. Thus, users managing the

access control are forced to deal with the low-level ACEs, their ordering in an ACL, and the formulas for resolving overlapping ACEs.

In this analysis of the decision process we did not consider administrative privileges that can change the ACL (e.g., the *write-acl* privilege in the WebDAV ACL model or *change permissions* permission in the Windows NTFS ACL model). Any user who has the *write-acl* privilege can modify the ACL to grant anyone (including him- or herself) any privilege. Taking this into account, any "deny" intention must resolve another intention, "principal X must not have the *write-acl* privilege on object Z", before resolving the intention *G2*; otherwise principal X may still grant him- or herself privilege Y at any time by modifying the ACL.

It should be clear from examining these two flowcharts that the process of deciding how to implement a particular intention is neither simple in terms of algorithmic complexity we defined above nor in terms of comprehension and examination of the system state. A user must know what questions to ask of the ACL system and in what order, and then may be forced to make some difficult decisions to achieve the desired goal. Thus, the task of formulating changes to an ACL to achieve even these simple intentions is far from trivial.

Further, in addition to the difficulties the user may have in resolving the intention, the lack of system support for answering these queries (e.g., an interface providing ready access to needed information), or of user knowledge for properly asking and resolving these queries, may lead to user errors. There are potential risks or errors associated with each query. Errors can be classified into two types: errors of commission due to the user establishing a wrong answer to a query; and errors of omission due to the user failing to make a query. For example, let us consider Q1. If the user gets the wrong answer "yes" to Q1 due to either lack of information or mistaken evaluation, he may falsely conclude that the intention is achieved and that nothing needs to be done (error of commission). If the user does not know to ask G4, he may falsely conclude that the intention cannot be achieved in any way (error of omission). Thus, tools designed to reduce the workload of users should not only make access control systems easier to use but also reduce the occurrence of these user errors.

The terms "Gulf of Evaluation" and "Gulf of Execution" introduced by Norman [12] are a good reference in the discussion of security usability. In conclusion, the structure of the algorithm described above represents the complexity of the "gulf of execution" for any interface to directly manipulate ACLs. The queries derived from the decision points represent a minimal set of states to be presented by an interface to cover the "gulf of evaluation".

Finally, this analysis reveals three features of the problem that will be important in designing systems to interact with this model: side effects, conflicts and user modeling decisions.

2.2.3 Side Effects

Any time a simple intention "principal X must have privilege Y on object Z" is resolved by the addition of principal X to some group G, it is likely that this action will have side effects. Suddenly, principal X may be granted (or denied) other privileges that have been associated with group G. At a minimum, the user should be notified of these side effects. Similarly, side effects may arise if the intention is resolved by the removal of principal X from some group G. In addition, if ACL inheritance is enabled and the intention G1 is resolved by modifying the inherited ACL, side effects may arise on any resource that inherits from the modified ACL.

There are, however, two other possible complications. It is possible that one of these side effects changes the results of a query in one of the previous branch points in the flowchart. In this case, we do have a loop and must go back and re-examine these points. The other possibility is that there may be a number of different groups with privilege Y on object Z that the user could add principal X to. In that case, we must consider that there are some decision points with multiple branches leading out of them, and we must allow the user to decide which of these would be preferable. We call these decision points *modeling decisions*.

2.2.4 Conflicts

Of course, there is no reason to have any sort of access control system in a single-user environment, so we should not conclude our analysis without due consideration of other users. Let us assume that all of the users of the WebDAV repository are basing their ACL implementations on similar intentional processes. As we pointed out above, it can be important to detect situations in which it is likely that two users' intentions are in conflict (i.e., one user acts to ensure a privilege Y for principal X on object Z and another user acts to deny that same privilege for principal X). In addition, because of the privilege hierarchy, two users' intentions may be in partial conflict, in which case the privileges in these two intentions do not exactly match but overlap.

What sort of consequences should there be when such a conflict is detected? We could adopt a "most-recent-change-wins" policy, in which case there is an obligation to at least inform the source of the conflicting ACE that his or her intention has been superseded. An alternative would be to disallow such supersessions and instead initiate a request to the source of the first entry that they retract their intention or modify it to allow an exception. This step could be augmented by an automatic process to suggest alternatives for resolving the conflict, a process that would likely have to be analyzed as we have done above.

Note that there are two kinds of conflict: the conflict between the ACEs and the conflict between the user intentions. Users care about the conflicts between intentions. However, even when two intentions do not conflict, it is possible that the resulting ACEs are in conflict, if the implementation of the previous intention causes some side-effects which contradict with the latter intention. We will not explore the issue here, but leave it for future research.

2.2.5 Modeling Decisions

The above example exposed one kind of modeling decision, a choice of which of a set of possible groups the user might expand or contract membership to in order to resolve an intention. It is likely that a combination of the "identities" of the groups and the side effects resulting from this change will determine which group the user will choose. A user may want to know what the options are to make such a decision, or simply choose the option with the fewest side effects. It may not be unreasonable to consider this difference to be subject to a user preference.

Another kind of modeling decision may arise in the case where

the user identifies a set of principals in his initial intention. In that case, we have the modeling decisions shown in Figure 3. We may be able to simply add individual grants directly, create a new group for those principals and add a grant to that group, or even add all of the principals to an existing group with the desired privilege. Of course, this last choice may cause side effects.

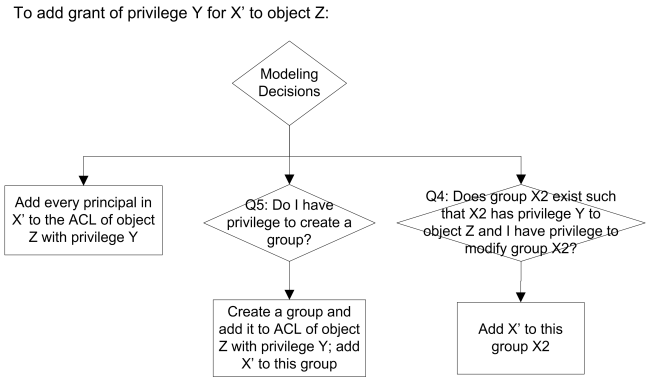

Figure 3. Example of a user's modeling decisions for *G1*

2.3 Discussions

2.3.1 Other Intentions

Consider also the "special" reflexive intention "I should have privilege Y on object Z". At first glance, this would seem to be just a variation on the intention G1; however it has very different implications. First, if it is possible for me to grant myself privileges that I didn't previously have, then it could certainly be argued that the system state is insecure, since no user should be able to grant him- or herself previously unavailable privileges in a secure system. But with the mix of direct and indirect privilege granting that comes from the addition of groups, it may be difficult to maintain this level of security throughout. In addition, it is unavoidable in the Windows NTFS file system that I can grant myself any privilege if I am the owner, because the owner of a file or folder in the Windows NTFS system can always change permissions on it, regardless of existing permissions protecting the file or folder.

Even in a secure system state it may be useful to submit this intention, since a conflict identified by it will point to the reason why the user has been denied access [9]. A subsequent query could then be made: "Who can grant me access?" Of course, for the intentions discussed previously, limited access to the ACL state of the system was not a problem. For example, if the user cannot examine the membership or permissions of a group, then it is irrelevant to the intention of using that group to grant a privilege, since the user will not be able to modify its membership anyway. With the "Who can grant me access?" query, however, it may be necessary to be able to examine such states. In this case, a list of people to ask would not necessarily violate the information hiding that presumably has been done deliberately if it is only their identity that is being revealed. In fact, if the intentions are transferable, then the user could effectively request access by passing on a description of his intention to someone capable of granting it and then allow them to decide whether and how to do so.

2.3.2 More Complex Cases

To this point, we have considered only simple intentions. Even for these simple cases, the set of queries that a user needs and the system should support (Q1-Q7) are neither simple nor small in number. They illustrate that even for simple intentions, the cognitive workload of users in constructing sequential and conditional queries and interacting with the system for modeling decisions is relatively heavy compared to the simplicity of the intentions. Consider complex intentions such as the following: "All the members in the groups to which my wife or I belongs except Bob have privilege Y on object Z." Clearly it is not an easy task for the user to transfer this intention to actual access control list settings. In the worst case, there may be ambiguities in a user's intentions. The system may need to detect such ambiguities and provide feedback to help the user make his or her intentions clear.

Some access control systems may group the privileges in a privilege hierarchy. For example, the privilege *read* may contain a sub-privilege *read-acl* that is the privilege to read the ACL. Surely it will complicate the decision process of the user to fulfill his/her goals, in either assessing the system state or planning the implementation.

Further, the privileges we have discussed so far are low-level privileges that are implementation-specific and can be set directly in the ACEs. However, the user, especially the end-user, may often have high-level privileges in mind that are not necessarily the same as the low-level privileges. In this case, the user has to understand the semantics of these low-level privileges. He or she then has to translate the high-level privileges to these low-level privileges before he or she can conduct a decision process such as the one described above for determining how to implement his or her intention. Therefore, the user has to construct a mapping between the high-level privileges and the actual low-level privileges which can be applied in the system. It is clear that such translations increase the user's cognitive workload further; a system that can perform these translations automatically or semi-automatically will be more usable.

The WebDAV access control specification indicates that the server implementation may support some ACL restrictions. For example, the "deny-before-grant" restriction specifies that all non-inherited <deny> ACEs must precede all non-inherited <grant> ACEs. Actually, this is a restriction employed by the Windows NTFS. The "grant-only" restriction indicates that ACEs with deny clauses are not allowed. In this case, if the user wants to deny principal X the privilege Y, and principal X holds the privilege Y by both an ACE granting him this privilege and an ACE granting a group of which he is a member this privilege, just removing the former will not fulfill his intention. In addition, the WebDAV access control specification also introduces some ACL preconditions. For instance, the "no-inherited-ace-conflict" specifies that the ACEs to be set must not conflict with the inherited ACEs on a resource. If such ACL restrictions or preconditions must be enforced in the access control systems concerned, the intentional analysis will be more complex.

2.3.3 Beyond the Algorithmic Analysis

In the algorithmic analysis above, we deliberately chose simple cases. Most other access control systems (e.g., policy-based, RBAC-based) are at least as complicated, and thus likely more conceptually complex and potentially "unusable". The first

finding we can naturally derive from the above analysis is that any user interface that lets users manipulate the embedded access control mechanism should provide enough and needed information to users so that they can fulfill their tasks quickly and accurately. For example, this analysis has exposed the views of system state necessary to effectively manipulate the ACL in terms of the queries Q1-Q7. This claim is supported by the work of Maxion and Reeder [11]. Their Salmon interface can provide answers to queries Q1, Q2, Q3, Q6, partial answers to Q4 and Q7, but no answer to Q5. One consequence of the analysis would thus be to refine the Salmon interface to better support the evaluation of Q4, Q5 and Q7.

However, we suggest that manipulating the ACL directly is not the right goal for a user interface design. Just visibility (i.e., good feedback) may not be going to work. Consider the complexity revealed by the above analysis. The factors, including the ordering of ACEs and its restrictions (e.g., deny-before-grant, no-inherited-ace-conflict), indirections by group membership and ACL inheritance, the privilege hierarchy, and administrative privileges, all complicate the decision processes for end-users. For an interface representing the interactions of manipulating the ACL, it is necessary to expose such complexity to end-users. The procedures and reasoning that are necessary to determine how to manipulate the ACL for fulfilling the goals are left to the users. If the system does not help/guide end-users to fulfill their goals according to the above analysis, end-users have to learn and retain the procedures by themselves while they are only interested in the system effects or results - effective privileges.

Therefore, we suggest that the next step is to consider the design of systems that limit the need for the user to be exposed to such complexity. The above analysis has shown that the translation between the ACL and user goals is just algorithmic and predictable. Therefore, we may design systems that can support the expression of user intentions and then resolve these intentions on the user's behalf, or at least provide significant offloading of the cognitive workload, which produces usable interfaces for these access control systems. Such systems present a higher level of abstraction and can be regarded as an incremental compiler that compiles down the high-level language (i.e., goals) to the low-level assembly language (i.e., ACL implementation). More specifically, we need support of visibility and manipulation in terms of effective access control as well as reasoning support to deal with indirection issues and limited manipulation. In this sense, the Salmon interface [11] can be taken as a front end of such systems while the input is changed to effective permissions instead of stated permissions. We will refer to such systems as *Intentional Access Management* systems and discuss them in the next section.

3. INTENTIONAL ACCESS MANAGEMENT

3.1 Design Principles

It must be emphasized that users of privacy/security systems view them as means to an end and not an end in themselves [15]. The systems are always peripheral to a user's primary task. Therefore, users of security management systems have little or no interest in solving "puzzles" to be able to use them. The analysis in Section 2 exposed some of these "puzzles". Reluctance to solve them is especially true of end-users, since they usually have limited expertise and interest in security systems compared to administrators. In this context we wish to extend the design principles of *Clarity* and *Visibility* from Yee [17] and recast them into a form specific to the analysis we have just completed. In general, for systems that involve risky or critical decisions (such as security systems), we claim that:

> *User decisions should be made/requested in an environment where*
>
> 1. *The user has access to essential information needed to make the decision reliably; and*
> 2. *The system should be responsible for predicting and presenting such information when it can.*

The translation of these principles into a system that takes user intentions as input and attempts to resolve them in a way that formulates access control rules that fulfill these intentions will result in what we refer to as an *Intentional Access Management* system.

We define an intentional access management system (IAM) as any system in which the following are true:

1. The user initiates interaction with the system by expressing an intention in terms of an output constraint on the access control system;
2. The system translates these intentions into implementation;
3. The system follows Yee's principles of clarity and visibility in informing the user of the consequences of actions not directly implied by their intentions; and
4. The system informs the user of modeling variations as well as detected ambiguities and conflicts in intentions.

We identify three possible levels of support for intentional access management: the *wizard* model, the full IAM model, and the multi-backend IAM model.

3.2 The IAM Wizard

One of the simplest ways to achieve the IAM model is to allow a user to express an intention and then use a "wizard" to walk the user through the process of creating an implementation of that requirement. This approach is similar to the various wizards used to allow some end-user system management in Windows environments. Wizards essentially walk a user through the decision process and request him or her to make choices when necessary.

Such a wizard for access management can be implemented by simply following the flowchart for implementing a goal and changing the "wizard window" whenever a modeling decision that refines the goal must be made. This window should at minimum provide access to some representation of the side effects of the modeling decisions (*visibility)*, and some description of the conflicts found as well as possible ways to resolve them. It must also make clear what the actions taken were and what the side effects of these actions are (*clarity).*

In addition, the interruption inherent in the wizard model may be mitigated by adopting the Surprise-Explain-Reward strategy [16].

One problem with the pure wizard approach, however, is in detecting and resolving conflicts. This model assumes that

intentions are expressed incrementally and that incremental changes are made to the system state to fulfill these intentions. Ultimately, the current system state is derived from a series of intentions expressed by a variety of users. Without some record of who made a change and what their intention was in doing so, it becomes very difficult for the system to do more than just notice that conflicts exist (e.g., between an existing grant and an intent to deny a privilege). If we wish to be able to actually resolve these conflicts, we need a system that maintains a record of intentions on a per-user basis and relates these to the current system state.

3.3 Full IAM

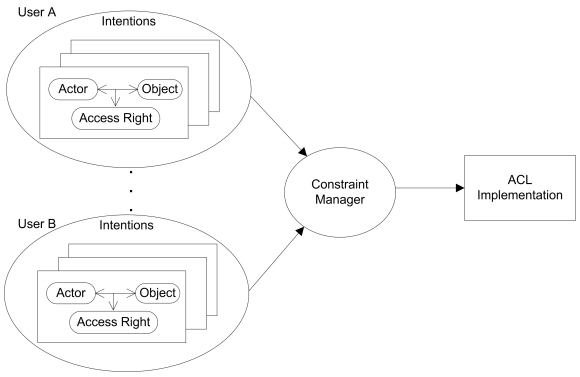

Figure 4. Full intentional access management (IAM) model.

Figure 4 presents a model that can deal with conflicts intelligently. It shows a full intentional access management system that, in addition to the requirements for basic IAM, includes:

1. the maintenance of intentions for each user;
2. the ability to retract previous intentions (like "undo" in direct manipulation);
3. the maintenance of connections between intentions and implementation actions; and
4. the management of conflicts by initiating user interactions to resolve conflicts.

As depicted in Figure 4, in an intention management system built upon this model, users can present their intentions based on their own conceptual access control model. The Constraint Manager module will then interpret these intentions and transfer them into appropriate access control implementation.

The Constraint Manager deals with constraint satisfaction and conflict resolution. Some set of intentions/goals {G} exists, and an access control system is configured to fulfill them. Constraint satisfaction creates and maintains the dependency D(S, G) on an access control statement S that helps to achieve an intention/goal G (i.e., is produced as a result of introducing intention/goal G).

A conflict occurs when a new intention/goal G' is introduced that contradicts an existing intention/goal G. If G' and G are both generated by the same user, then G' has priority over G, but the user should be notified. If user U' introduced G' and user U introduced G, then we have an inter-user conflict and need some strategy of notification and conflict resolution. Note that here we do not introduce any new complexity to access control systems, but expose the inherent complexity of the systems in the multi-user environment.

3.4 Multi-Backend IAM

Finally, it becomes feasible to expand this model to one in which there may be a variety of different implementation models available as the backend, depending on the storage repository being used. If done correctly, the intentional model may be able to present the user with a unified conceptual model and interface independent of the means being used to implement the access control. Thus through the multi-backend IAM, end-users can control multiple systems embedding various access control mechanisms in a unified way without learning them separately. At the same time administrators can still manage individual systems through their traditional way. Of course this will raise a challenging issue of mapping the administrators' administrative actions to their intentions, if we want to detect and resolve the conflicts between the intentions of administrators and end-users.

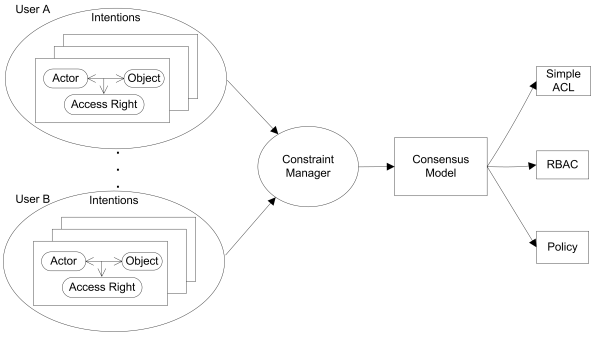

Figure 5. Multi-Backend IAM model.

To achieve such security interoperability, a Consensus Model needs to be constructed as a mediator between the Constraint Manager and the security backend implementations, as shown in Figure 5. The consensus model will abstract access control out of the concrete implementations such as simple ACL, RBAC-based, and/or policy-based systems. In this way, a general constraint manager can be maintained no matter what access control mechanism is implemented in the backend.

Validation of the proposed intentional access management models is a concern. However, we can confidently claim that the proposed models can be implemented, because the basic intentions G1 ("Principal X must have privilege Y on object Z") and G2 ("Principal X must not have privilege Y on object Z") represent the semantics embedded in the access matrix [10] lying at the heart of most access control models. In essence then, the goal statements are simply modeling the same access matrix as traditional access control systems, except as constraints on the effective privileges. Absent conflicts then (which we argued should primarily be resolved outside the access control system), both IAM and traditional access control are constraining the same resource. Therefore, if an access control system is sufficiently powerful to describe any potential access matrix (and the intentional model clearly is), the problem of deriving an access control implementation from a set of intentional constraints is simply algorithmic

4. IMPLEMENTATION

To demonstrate the applicability and inherent usability of the intentional access management designs, we have applied the proposed principles and models to an access management

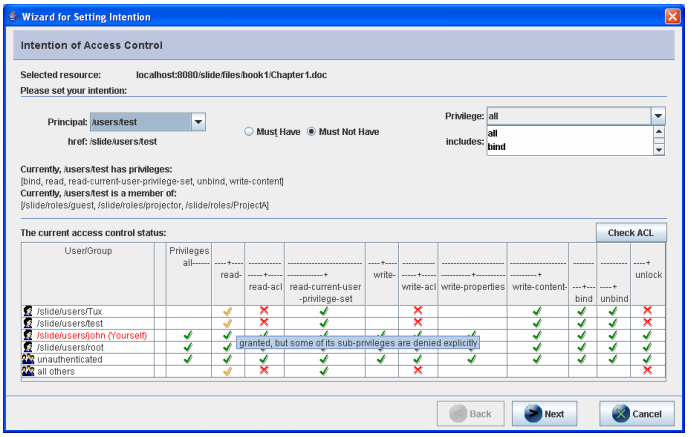

Figure 6. A screenshot of DialogA obtaining the intention from the user and showing the current access control state

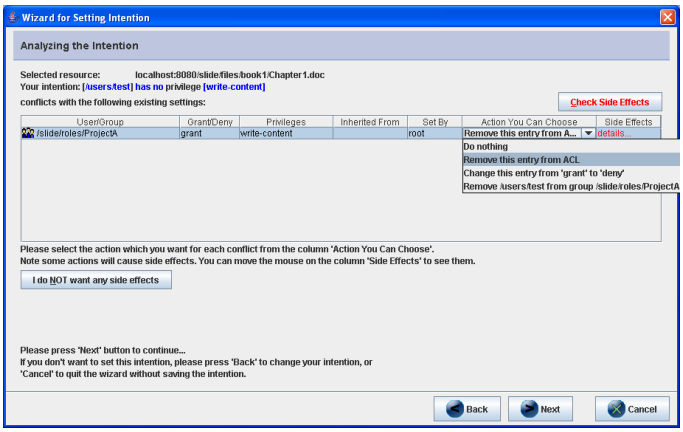

Figure 7. A screenshot of DialogB showing the conflicts and modeling decisions

prototype for WebDAV. Any server supporting WebDAV may be used for this system; we chose Slide server [8]. As for the client, we chose DAV Explorer [5], whose user interface is similar in look and functionality to the Explorer program in Windows. In the current implementation, an IAM wizard was built into the DAV Explorer. It supports basic intentions of who must / must not have what privileges on the object.

We based the program flow of the IAM wizard on the decision processes derived from the intentional analysis described in Section 2, but all reasoning work is now performed by the IAM instead of the user. To fulfill the user's intention, the system state will be changed by adding/removing/modifying ACEs or adding/removing principals into/from groups, automatically or according to the modeling decision the user chooses. Three main dialog windows (DialogA, DialogB, DialogC) were designed to accomplish this flow.

DialogA (Figure 6) is split into two panes, upper and lower. The upper pane lets the user specify his or her intention regarding who must/must not have what privilege. Once a principal is selected, the pane also shows the privileges that principal currently has, and the groups that principal is a member of. In addition, the lower pane in the dialog window shows the current access control status by listing the effective privileges for each principal. Such information helps the user to construct a clear and correct view of the system state. In addition, tool tips and labels are extensively used to present explanations when the mouse pauses over preset UI components.

The content of DialogB may change according to different branches in the program flow. When the system detects that the intention conflicts with existing effective ACEs (those not preceded by relevant ACEs), this dialog will present all the conflicting ACEs to the user with the information of who set those ACEs. If the user has the privilege to modify the ACL, it will also provide possible solutions (modeling decisions) for the user to resolve the conflicts, as shown in Figure 7. Of course, some solutions may have side effects. The conflicting ACE will be highlighted if the user chooses the solution that has side effects, and then the user can click to check the side effects. By default the system will choose the solution without side effects. The user can always click the "I do NOT want any side effects" button to reset to the default solution.

If the user does not have the privilege to modify the ACL, the IAM system will automatically search for alternatives (e.g., adding the specified principal to some group) and present them in DialogB for the user to choose. Also, the user can check the side effects associated with each solution.

DialogC is the final window showing the task result. If the intention cannot be fulfilled, it will display error information and explanations. If the intention is successfully fulfilled, it will list all the actions the system has taken, in order. The user can also check current system state to verify the fulfillment.

5. USER STUDY

To evaluate the usability of the proposed IAM system, a carefully designed laboratory user study was conducted. Two modules for managing WebDAV access control were compared: the simple ACL editor like module in the original DAV Explorer (referred to as ACL Editor in the study), and the IAM wizard, a new module for managing access, implemented based on the IAM models. The current user study is preliminary. We do not claim that the IAM is the best approach to making access control usable for end-users. The goal of this study is to demonstrate that the proposed design is feasible and usable for end-users while it doesn't upset user expectations or cause confusion. In other words, the study was designed to expose failures in the conceptual model of the user interaction rather than to test specific claims of its efficiency.

5.1 Study Design

5.1.1 Participants

Ten people participated in the study. Participants were recruited randomly and had various backgrounds from business to engineering. All used computers at least a few times a week. Eight reported having some experience setting file permissions on Windows or another operating system, while two reported having no experience setting file permissions whatsoever. Two reported they were averagely familiar with the ACL and how it is evaluated prior to the study, and five reported knowing a little about the ACL, while three did not know the ACL at all. None knew how ACLs are implemented in WebDAV before the study.

5.1.2 Task Descriptions

To simulate real access management conditions, a hypothetical scenario was designed in which the participants at different sites collaborated to jointly author documents which were stored on a WebDAV server, and had to restrict access to these shared files. The hypothetical collaborative environment was created and populated with individual users, groups, files, and folders on the WebDAV server. The environment included 8 individual users, plus one user named John who represented the participant her/himself. The environment also includes 7 groups. No group contained another group as a member.

Two sets of tasks were given to each participant. These two sets were identical except that they targeted different resources. Each set of tasks included one training task which gave participants experience with the module used, and four other tasks (Task1, Task2, Task3, and Task4). Due to the page limitation, we do not list the detailed task statements here.

The training task simply required the participant to add an ACE granting *users/Jack* the privilege *write-content*, and to find that all users had had the privilege to read the file.

Task1 was introduced to check if the participant could determine that the task requirements were already fulfilled and no action was needed.

The task statement for Task2 was identical to that for Task3 except for the names of specific files and users. They asked the participant to configure the system so that the specific user can read but cannot change the content of one file. However, the tasks differed in the way they were initialized. In both of these tasks, there was one group that was already on the ACL for the file, and the target user was a member of that group. However, in Task2, the user *users/test* had only inherited the *write-content* (the privilege to change file content) privilege from group ProjectA, while in Task3, the user *users/projector* had inherited the *write* privilege from group ProjectB which contained both *write-content* and *write-acl* (the privilege to modify the ACL) privileges.

The simple solution to Task2 was to add an ACE denying *users/test* the *write-content* privilege; he already had the privilege to read the file content. However, this simple solution could not work for Task3, since *users/projector* had inherited the *write-acl* privilege from group ProjectB. If *users/projector* was denied the *write-content* privilege, but not explicitly denied the *write-acl* privilege, he would have been able to restore his *write-content* privilege. The task statement presented to users did not mention this nuance; it was left to users to decide that *users/projector*'s *write-acl* privilege had to be removed.

Note that Task2 and Task3 are similar to the Wesley and Jack tasks in Maxion and Reeder's study [11] on the Windows XP File Permissions interface and their Salmon interface.

The purpose of Task4 was to check if participants could find an alternative way to grant privileges (adding user *users/john2* to group *roles/ProjectC*) when they did not have the privilege to modify the ACL.

In order to collect their feedback on the IAM wizard, a short interview with each participant was performed after the participant finished all the tasks.

5.1.3 Procedure

Participants were asked to use the ACL Editor to fulfill one of the two sets of access management tasks and use the IAM wizard to fulfill the other set of tasks. For Task1, Task2, and Task3, eight used the ACL Editor first and then the IAM wizard, and two used them in the reversed order. For Task4, they all used the ACL Editor first, and then the IAM wizard. Before participants used the ACL Editor to fulfill tasks, they were required to learn how the ACL is evaluated. Participants were told that they could "think aloud" throughout the course of the experiment. Participants were shown how to view WebDAV system users, groups, group memberships, and the owner of a resource. After each task was completed, participants were asked to rate their confidence on a 1-7 scale (7: very confident) that the task had been completed correctly.

5.2 Results

This section presents the results of the study, including speed, accuracy, user confidence and satisfaction for each of the two modules under scrutiny. The results show that the IAM wizard performed better than the ACL Editor did.

5.2.1 Speed

For accomplishing the tasks by using the ACL Editor, all participants had to first learn the ACL and how it is evaluated. The average time for them to understand the ACL evaluation is about 8 minutes. On the contrary, the two participants who fulfilled the tasks by using the IAM wizard first spent no time learning the ACL evaluation before the tasks but still got 100% accurate task completion.

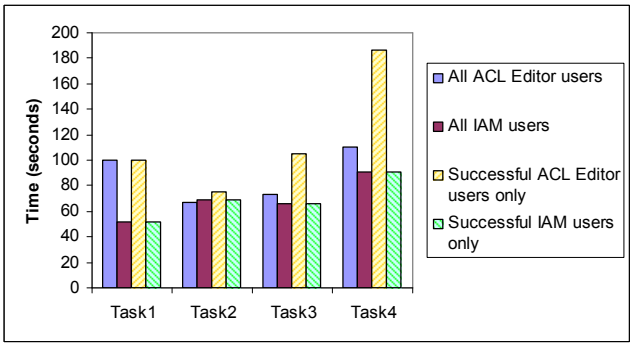

Figure 8. Average time to complete Task1 - Task4

Figure 8 illustrates the average task completion times for each of the two modules and four tasks. The solid bars show times for all participants, whether they succeeded or failed in the task; the striped bars show times only for participants who completed the tasks accurately. For using the IAM wizard, since all tasks were successful, the average times for all and for successful participants were the same for each task. Note that the difference between Task4's average completion times for all participants and only successful participants using the ACL Editor is large. For this task, 9 of 10 participants using the ACL Editor thought they could not complete the task and gave up while one went on to check the "Group Manager" we developed for the DAV Explorer to manage group membership.

These results show that those completed the tasks successfully took less time using the IAM wizard than using the ACL Editor. For Task2 and Task3, the difference between times for the two

modules is not statistically significant (one-sided Welch's *t*-test for Task2: $t = 0.3511$, df = 7, $p = 0.3679$; for Task3: $t = 1.6482$, df = 2, $p = 0.1205$). However, for Task1, successful users using the IAM wizard spent, on average, significantly less time than the successful users using the ACL Editor did. A one-sided Welch's *t*-test showed this difference to be statistically significant at the 0.05 level ($t = 3.3789$, df = 18, $p = 0.0017$). This is because many users using the ACL Editor still added new ACEs into the ACL although the task requirements had already been fulfilled, while the IAM wizard directly told them the fact and then no action was needed. For Task4, the time difference is also significant. The only one who successfully completed this task using the ACL Editor took 187 seconds (he first used the ACL Editor and then the Group Manager), while the users using the IAM wizard took less time (Mean = 91.6, Standard Deviation (SD) = 31.3).

5.2.2 Accuracy

Table 1 shows the percentages of participants who successfully (accurately) completed the tasks by using the ACL Editor and the IAM wizard. The IAM wizard outperformed the ACL Editor on all tasks. Especially, for Task4, using the ACL Editor, only 1 participant who remembered his prior experience of adding users to a group to get privileges completed the task successfully, while all participants completed the task successfully by using the IAM wizard.

Table 1. Percent of accurate completions for the four tasks by using the ACL Editor and the IAM wizard

	Using ACL Editor (%)	Using IAM wizard (%)
Task1	100	100
Task2	70	100
Task3	30	100
Task4	10	100

Note that by using the IAM wizard, more users correctly completed the tasks while taking less time on average, suggesting that IAM's accuracy gains were not due simply to a speed-accuracy tradeoff.

As indicated above, Task2 and Task3 are similar to the Wesley and Jack tasks in Maxion and Reeder's study [11]. In that study, their proposed Salmon interface which only provided needed information to the user for setting file permissions recorded 83% and 100% accurate task completions for the Jack and Wesley tasks, respectively. The IAM wizard with both 100% accurate task completions demonstrates superior performance.

Actually, these results are natural and predictable, because the IAM is designed to directly accommodate user goals. It eliminates human errors in the goal implementation. In such a system, the only source of error is in expressing the intention/goal. For example, it is possible that the representation of some complex intentions in the system does not match the user's mental model, and this may lead to confusion and failure to express such intentions to the system. Therefore we need to study the users' real security intentions and represent them more accurately in the system. One useful work is to model user privacy needs for information sharing.

5.2.3 User Confidence and Satisfaction

The participants were asked to state their confidence in their work. Of course, they gave the highest confidence rating (i.e., 7) to all tasks when using the IAM wizard, because the system accomplished most work for them, and provided enough feedback so that they can check the results in the end. When using the ACL Editor, as shown in Table 2, their confidence was lower.

Table 2. Average confidence ratings for the four tasks by using the ACL Editor and the IAM wizard

	Using ACL Editor (Mean, SD)	Using IAM wizard (Mean, SD)
Task1	6.9, 0.3	7, 0
Task2	6.4, 0.7	7, 0
Task3	6.3, 0.8	7, 0
Task4	1.6, 1.8	7, 0

Note the mismatch between the high confidence and poor performance on Task 2 and Task 3 when using the ACL Editor. This is a recipe for security failures and frustration. Users think that they have correctly configured the security system to protect their resources, but they are mistaken. Moreover, it is unlikely that this mistake would be noticed until a security breach demonstrates that the system is misconfigured (assuming that they are made aware of the breach). Since the IAM design addresses both the gulf of execution and the gulf of evaluation, the user's confidence is a good match to goal success even though the system hides many of the details of the mechanism. In essence, because the system is careful to show the user the effects of their actions, the "magical" connection between the intention and the mechanism seems natural and predictable.

In the interview, all participants expressed their preference of the IAM wizard over the ACL Editor. Most of them explicitly indicated that this level of expressing goals seemed natural to them, and the hiding of the internal security mechanism (i.e., the ACL) did not confuse them, or upset their expectations. The most common feedback was that the new module was straightforward for accomplishing the tasks.

6. RELATED WORK

There are few studies on usability of access control mechanisms. The Adage project [19], the successor of MAP project [18], may be the largest effort to date to employ usability design techniques in developing an authorization service for distributed application. However, MAP and Adage were intended for use by professional system administrators who already possess a high level of expertise, and as such they did not address the problems posed in making security effectively usable for a more general population of end-users.

The work most directly related to the topic of this paper is the study performed by Maxion and Reeder on the Windows NTFS file permissions model [11]. As described in Section 2, their new interface Salmon can be seen as an interface that provides the user direct manipulation on the internal access control mechanism (i.e., the ACL) with useful feedback (e.g., displaying effective permissions). Salmon was designed to provide an accurate, clear and salient external representation of the information needed to achieve the user's primary goal. However, the user still has to formulate subgoals from his or her primary goal according to the information provided by Salmon and determine by him- or herself how to implement these subgoals. Considering the low interest and expertise of end-users, it is usually not an easy task for them, even with useful feedback provided by the interface.

Another recent related work is the study by Brostoff et al. [3] on improving the usability of their PERMIS system for writing authorization policies for e-Scientists, their target end-users. Through user trials, they identified two fundamental problems: lack of understanding what the policy components are and lack of understanding of the underlying policy paradigm. To address these problems, they revised the user interface (UI) labels to better describe access policy components, and used instructional text in the GUI and UI behaviour to shape users' models of the policy paradigm. However, like Salmon, as they indicated, thorny problems still remain - users have to work out how to do what they now know needs doing. Our IAM models are designed to tackle such problems, and their work can be a good complement.

Some researchers have suggested a range of design principles for building systems that are both secure and usable [2] [17] [18]. For example, Yee [17] summarized ten design principles for secure interaction design which include *Clarity*, *Visibility* and *Expressiveness*. However, we argue that although these principles are desirable in designing security systems, their application may not guarantee the usability of the designed systems. Two problems exist: (1) a system so designed may be perfectly usable for one group of users and opaque and unusable for a different group (e.g., system administrators vs. end-users), and (2) these guidelines do nothing to address the gap in reasoning that could allow a user to determine what changes need to be made to a system's state to achieve a desired goal.

7. CONCLUSION

Through a thorough analysis of the WebDAV access control list model, we find that the cognitive workload required for even simple access management tasks is too heavy for end-users. To alleviate the conceptual complexity and make the management task easier for end-users, we introduce the concept of intentional access management (IAM). A set of design principles and three models for supporting the IAM are proposed. We have implemented an IAM wizard to manage ACLs in WebDAV, and performed a usability study to confirm that our approach is effective and usable. This research is only in its early stage. To demonstrate the high potential of IAM for usable access control, the full IAM and multi-backend IAM models need to be developed. In addition to the resolution of conflicts, and the presentation and exploration of side effects, such future IAM systems should support interactions with multiple systems and with system administrative actions.

8. REFERENCES

[1] Adams, A. and Sasse, M. A. Users are not the enemy. *Comm. ACM*, vol. 42, no. 12, 1999, 41-46.

[2] Balfanz, D., Durfee, G., Smetters, D.K., and Grinter, R.E. In search of usable security: five lessons from the field. *IEEE Security & Privacy Magazine*, vol. 2, no. 5, 2004, 19-24.

[3] Brostoff, S., Sasse, M. A., Chadwick, D., Cunningham, J., Mbanaso, U., and Otenko, S. R-What? Development of a role-based access control (RBAC) policy-writing tool for e-scientists. *Software: Practice and Experience*, vol. 35, no. 9, July 2005, 835-856.

[4] Clemm, G., Reschke, J., Sedlar, E., and Whitehead, J. *Web Distributed Authoring and Versioning (WebDAV) Access Control Protocol*. RFC 3744. May 2004. http://www.ietf.org/rfc/rfc3744.txt.

[5] DAV Explorer. http://www.ics.uci.edu/~webdav/

[6] Dussault, L. WebDAV benefits for the enterprise and its denizens. *DM Direct Newsletter*, Dmreview.com, June 27, 2003. http://www.dmreview.com/article_sub.cfm?articleID=6971

[7] Goland, Y., Whitehead, E., Faizi, A., Carter, S.R., and Jensen, D. *HTTP Extensions for Distributed Authoring -- WEBDAV*. RFC 2518. Feb. 1999. http://www.ietf.org/rfc/rfc2518.txt.

[8] Jakarta Slide project. http://jakarta.apache.org/slide/

[9] Kapadia, A., Sampemane, G., and Campbell, R. Know why your access was denied: regulating feedback for usable security. In *Proceedings of the 11th ACM conference on Computers and Communications Security (CCS)*, Washington DC, Oct 25-29 2004.

[10] Lampson, B. Protection. In *Proceedings of the Fifth Princeton Symposium on Information Sciences and Systems*, Princeton University, March 1971, 437-443.

[11] Maxion, R.A. and Reeder, R.W. Improving user-interface dependability through mitigation of human error. *International Journal of Human-Computer Studies*, vol. 63, 2005, 23-50.

[12] Norman, D. *The Design of Everyday Things*. Basic Books, New York, 2002.

[13] Sandhu, R.S., Coyne, E.J., Feinstein, H.L., and Youman, C.E. Role-based access control models. *IEEE Computer*, vol. 29, no. 2, February 1996, 38-47.

[14] Sheehan, K. Towards a typology of Internet users and online privacy concerns. *The Information Society*, vol. 18, 2002, 21-23.

[15] Whitten, A. and Tygar, J.D. Why Johnny can't encrypt: a usability evaluation of PGP 5.0. In *Proceedings of 8th Usenix Security Symposium*, Usenix Assoc., 1999, 169-184.

[16] Wilson, A., Burnett, M., Beckwith, L., Granatir, O., Casburn, L., Cook, C., Durham, M., and Rothermel, G. Harnessing curiosity to increase correctness in end-user programming. In *Proceedings of ACM Conference on Human Factors in Computing Systems*, Ft. Lauderdale, FL, April 2003.

[17] Yee, K.-P. User interaction design for secure systems. In *Proceedings of 4th International Conference on Information and Communications Security*, Deng R. et al., eds., LNCS 2513 Springer, 2002, 278-290; http://zesty.ca/sid.

[18] Zurko, M. E. and Simon, R. T. User-centered security. In *Proceedings of the ACM New Security Paradigms Workshop*, 1996, 27-33.

[19] Zurko, M. E., Simon, R., and Sanfilippo, T. A user-centered, modular authorization service built on an RBAC foundation. In *Proceedings of the IEEE Symposium on Security and Privacy*, 9-12 May 1999, 57-71.