SOFTWARE-PRACTICE AND EXPERIENCE
Softw. Prac. Exper. 2000:00:1-7

"R-What?" Development of a Role-Based Access Control (RBAC) Policy-Writing Tool for e-Scientists

Sacha Brostoff[a], M. Angela Sasse[*†], and
David Chadwick[b], James Cunningham, Uche Mbanaso, Sassa Otenko[c]

[a] Department of Computer Science, UCL (University College London)

[b] Computing Laboratory, University of Kent

[c] Information Systems Institute, University of Salford

SUMMARY

A lightweight role-based access control policy authoring tool was developed for e-Scientists, a community where access policies have to be implemented for an increasingly heterogeneous group of local and remote users. Two fundamental problems were identified (1) lack of understanding what the policy components are (i.e. how authorization policies are structured), and (2) lack of understanding of the underlying policy paradigm (i.e. what should go into the policy, and what should be left out). Conceptual design (CD) techniques were used to revise the user interface (UI) labels so that e-Scientists and developers were better able to describe access policy components from labels, and match labels with components ($t=6.28$, $df=7$, $p=.000$ two tailed). CD, instructional text, bubble help, UI behaviour and alert boxes were used to shape users' models of the policy paradigm. The final prototype improved users' efficiency and effectiveness by: more than doubling the speed with which expert users could write authorization policies; and facilitating users without specialist security knowledge to overcome the policy paradigm and components problems, enabling them to complete 80% of basic and 75% of advanced authorization policy writing tasks in a usability trial.

1 Introduction

This paper presents a case study in which usability techniques were applied to make a tool for writing Role-Based Access Control (RBAC) policies more accessible to e-Scientists. e-Scientists share their equipment, data, and software with increasingly large groups of geographically distributed collaborators, and consequently they face the need to protect the confidentiality, integrity and availability of their resources from unauthorized access and malicious attacks.

[*] Correspondence to : M. Angela Sasse, Department of Computer Science, UCL (University College London)

[†] Email: a.sasse@cs.ucl.ac.uk

Grids share computer resources to an unprecedented level. As the number of users and the connectedness of e-Science resources increases, so does the importance and difficulty of granting appropriate authorization and access to these resources. An easy to use tool is needed to support the granting of access rights. Because a Grid environment is such a severe test for writing correct authorization policies, e-Scientists' need for a usable authorization policy management tool is particularly great.

This paper describes part of the effort to provide such a tool for the e-Scientist – a general tool to create and edit authorization policies that can subsequently be enforced by the PERMIS authorization infrastructure [4]. Most implementations of role based access controls are designed to be used by system administrators – highly technical people with expertise in access control, and a desire to read the manual [30]. This paper is an attempt at making a state-of-the-art, flexible authorization infrastructure available to e-Scientists – users who are less technically capable than the system administrators who usually employ role based access controls, and far less motivated by security.

The rest of this paper is structured as follows. We start by introducing the security concepts and technologies that are relevant to this paper i.e. authorization, role based access controls, and the PERMIS authorization infrastructure. We then review previous literature in the area of security and usability before presenting our case study findings. Stages of our tool's development, focusing on usability issues, are outlined, along with each stage's methods, their results and subsequent diagnoses for improvements. We conclude with a discussion of our positive and negative experiences of introducing usability into the design process.

2 Security Concepts and Technologies

2.1 Authorization

Authorization is a critical part of security. Saltzer and Schroeder define authorization as "grant(ing) a principal access to certain information" [20]. Coming after identification (the user saying who she is) and authentication (verifying that the user is really who she says she is), authorization says what the user is allowed to do. Authorization mechanisms provide safeguards against benign users mistakenly accessing confidential information that they are not entitled to see, as well as against attackers attempting to hijack computing and information resources and using them for illegal purposes. (Note that no security mechanism is 100% secure against all forms of attack; we usually only provide cost effective security mechanisms commensurate with the risk exposure.)

Authorization infrastructures typically comprise two parts. Firstly, a mechanism for granting access rights to users, and secondly a mechanism for controlling or enforcing those rights (often called access control systems). Access control systems are most frequently part of a computer's operating system. They monitor any requests for access, for example to read, write, or change data. When a request to perform an operation on the computer is made, it is checked against a statement of policy saying who is authorized to do what, usually held in the form of an access control list. If the request passes the policy test, it is allowed. If it fails, the request is denied and the requested action is stopped before it can begin. Inserting entries into the access control list is the mechanism used to grant access rights to users. Traditionally each application and operating system has had its own proprietary authorization mechanism, implemented as an integral part of the application or operating system code. This leads to a large administrative burden, since administrators need to learn to use different software tools in order to manage each application, and need to configure different access rights into each piece of software. Furthermore it is difficult, if not impossible, to have a consistent authorization policy across all applications and operating systems since the underlying access control models and mechanisms vary from system to system. When one considers the large distributed nature of Grid computing, with the hundreds, if not thousands, of different systems that are involved, one can see that managing the authorization of Grid users across Grid applications becomes a huge burden on administrators. Reducing this burden is one of the primary objectives of the project that is reported here.

2.2 Role Based Access Controls

Role Based Access Controls (RBAC) are a type of access control that is designed to reduce the administrative burden of access control management. Early work by Sandhu and colleagues [21] led to the eventual publication of RBAC as a NIST standard [16]. The key feature of RBAC is that a level of indirection is introduced between users and their authorized actions on resources. This intermediate concept is termed a role. So whilst an access control list says which actions a user is allowed to perform on which resources, in RBAC, role assignments state which roles a user is allowed to assert or possess, whilst role specifications state which actions a role (holder) is authorized to perform on which resources. Access control management is significantly reduced because typically the number of roles in an organization is less than 4% of the number of employees [23].

2.3 The PERMIS Authorization System

PERMIS is a RBAC authorization system that uses X.509 attribute certificates (ACs) [10] to hold users' roles. X.509 attribute certificates are digitally signed by the role assigner so that they cannot be unknowingly tampered with, and thus can be transported safely across unsecured networks and stored in public repositories such as LDAP directory services. PERMIS has an access control decision engine that makes authorization decisions (granted or denied) on behalf of its clients (applications and operating systems) based on the roles belonging to a user, the action the user wishes to perform, and the current authorization policy. The PERMIS authorization policy states which managers are trusted to assign which roles to which users (this is called the role assignment policy - RAP), and which roles are authorized to perform which actions on which resources (this is called the target access policy - TAP). If a user presents a role that is not trusted by the RAP, it will be discarded by the decision engine. If a user requests access to a resource that is not authorized by the TAP, the request will be denied. The PERMIS default policy is deny everything except that which is specifically granted in the current authorization policy. Clearly it is important that the administrator of a PERMIS controlled resource, be it an application such as a database or an operating system resource such as a filestore, knows how to create correct PERMIS authorization policies so that only the correct set of users are given the correct set of access rights to the resources that are being protected. Hence it is vitally important that the PERMIS policy management tool is correctly designed so as to be user friendly and intuitive to its administrators.

2.4 e-Science and PERMIS

e-Science is a program and movement to facilitate academic and commercial research in the sciences through better use of existing computing resources. This usually involves combining the resources of many computers together so that they are presented to the e-Scientist as a single, much greater resource. Inherent in this program is the sharing of resources and data across administrative boundaries. Different departments within a university might pool their compute-farms, and different universities might then pool these combined pools into a much larger computational grid. For example, the GridPP project spans 19 institutions, and is aiming to combine over 4,000 CPUs [3]. Sharing across administrative boundaries leads to the forming of virtual organisations (VOs), where each institution administers its data or resources (it is unwilling to relinquish control over them, and may have legal duties not to) but allows limited access to other members of the VO. VOs such as these form a difficult test case for authorisation systems, particularly when their administration is being put in the hands of non-experts – the e-Scientists themselves.

PERMIS is an infrastructure that allows Role-Based Access Control to be implemented in e-Science systems, such as but not limited to computational Grids. PERMIS consists of an application gateway (the decision engine), surrounding the Grid node, which looks up users' X.509 attribute-certificates (ACs) in an LDAP directory in response to their requests for access. The ACs contain the users' roles, which are compared against the policy on the Grid node listing the permitted roles and their access rights, and a decision to grant or deny access is made. A check is also carried out that the entity that issued the role AC to the user was authorized to do so.

PERMIS simplifies access control for managers of e-Science nodes through the use of the role based access control paradigm - privileges are given to roles rather than individual users. Whilst the individuals who fulfill roles in an organization may come and go with relative frequency, the roles themselves are relatively unchanging. In this paradigm, the access control task is simplified to giving privileges to roles, with an additional job of distributing roles to users. The latter management task is further simplified by delegating the task of distribution of role ACs to named individuals, called *Sources of Authority* (*SOAs*). The manager of an e-Science node need only write the names of these *SOAs* into the access control policy at his/her node, and then (s)he can leave it up to these *SOAs* (managers) in other locations or institutions comprising the e-Science project to distribute the role ACs to their own users.

3 Previous Related Research

Very little research directly related to the topic of building a user friendly policy management tool is actually published. There are countless papers on access control models, including RBAC, countless papers on policy language design and policy related issues, a reasonable number of papers on role engineering and role lifecycles, but virtually nothing about the user centered design of a RBAC policy management tool. The first paper to be written on this topic was by Mary Ellen Zurko and colleagues [30] in 1999. They state,

"Saltzer and Schroeder saw usability as a fundamental security principle, on a par with principles such as least privilege and fail-safe defaults. While these latter principles have become standards in the security literature, the user-centered principle has been honored more in the breach. Our work is the first that we know of that explicitly places security and usability as peer goals of an effort, and employs specific techniques to assure the achievement of both goals".

Part of their objective was to produce an RBAC policy management GUI for the US DOD. They, like us, recognized that security non-experts (they termed them casual users) would comprise a significant proportion of the user community that would need to write RBAC policies. Unfortunately, after they had built their first prototype policy management GUI, they were only able to test it with five users who were *experienced in managing security for distributed applications*. Thus a significant proportion of their intended user base was never utilized during their build and testing phases, which rather detracts from their results. Even so, when asked to evaluate their GUI representations of concepts on a scale from obvious to confusing, they found that even these experienced users found the following concepts somewhat confusing: *actions, constraints, roles* and *labels*. They concluded that further study was still required. We agree. If even experienced users found their terms somewhat confusing, then what would the casual users have made of them? This is one of the questions that are answered in our current work. Our work therefore goes further than the work of Zurko et.al, in that casual users were the main set of users that we used to test the various versions of our policy management GUI. However, experienced users were used as well.

The next significant piece of research is by Ka-Ping Yee [28]. He proposes ten design principles for secure user interaction with systems. These are

(i) *Path of Least Resistance.* The most natural way to do any task should also be the most secure way.

(ii) *Appropriate Boundaries.* The interface should expose, and the system should enforce, distinctions between objects and between actions along boundaries that matter to the user.

(iii) *Explicit Authorization.* A user's authorities must only be provided to other actors as a result of an explicit user action that is understood to imply granting.

(iv) *Visibility.* The interface should allow the user to easily review any active actors and authority relationships that would affect security-relevant decisions.

(v) *Revocability.* The interface should allow the user to easily revoke authorities that the user has granted, wherever revocation is possible.

(vi) *Expected Ability.* The interface must not give the user the impression that it is possible to do something that cannot actually be done.

(vii) *Trusted Path.* The interface must provide an unspoofable and faithful communication channel between the user and any entity trusted to manipulate authorities on the user's behalf.

(viii) *Identifability.* The interface should enforce that distinct objects and distinct actions have unspoofably identifable and distinguishable representations.

(ix) *Expressiveness.* The interface should provide enough expressive power (a) to describe a safe security policy without undue difficulty; and (b) to allow users to express security policies in terms that fit their goals.

(x) *Clarity.* The effect of any security-relevant action must be clearly apparent to the user before the action is taken. [28]

We have used these principles throughout our current work, and believe that all ten have been fulfilled in our final policy management tool. However, we do not believe that usable security software can be guaranteed by following design heuristics alone, and that to do so could result in sub-optimal or even dangerous usability flaws. At least, some user testing must be conducted on the resulting design. In a more recent paper [29], Yee acknowledges that iterative design is essential to producing secure and usable software, but that few developers seem to incorporate security and usability testing throughout the design and build iterations. This has been one of the main objectives of the current project.

The final significant piece of research is by Balfanz et.al [2]. They decided to install a secure wireless LAN in PARC using PKI certificates. The user enrollment process took the installers, all security and PKI experts, a couple of minutes to complete. Needless to say they were very surprised when the average time taken by normal users (who were all computing professionals with advanced degrees, mostly PhDs) was over 2 hours. As a result, they document five lessons from the field, which are:

(i) *you can't retrofit usable security*, meaning security cannot be bolted on afterwards, it has to be built in from the start

(ii) *tools aren't solutions*, meaning mechanisms such as SSL are good building blocks, but we are still missing other usually higher level building blocks when constructing secure applications

(iii) *mind the upper layers*, meaning that security cannot only be handled in the lower layers. It must be built into the upper layers as well, which will require security to be implicit and much more user friendly.

(iv) *keep your customers satisfied*, meaning put users' needs first, not those of the developers or even of the security officer.

(v) *think locally, act locally,* meaning that if security mechanisms can be kept local to the current system, then co-ordination with external security mechanisms is not needed, which can significantly simplify the user interaction.

As a result of their and our own experiences, we can now add a sixth lesson from the field, which is *"security experts are mainly blind to normal users (lack of) ability to understand security concepts and terminology"*. The reason for this blindness is totally understandable. An expert who uses a particular jargon and set of concepts daily, and who is completely familiar with their semantics and nuances, is simply not able to see what other people cannot see. The evidence for this sixth lesson comes from various sources. Balfanz et.al report that when they told other security experts it was taking users over 2 hours to perform PKI enrollment, the experts did not believe them, to the point of even suggesting that their empirical results must be incorrect. In our own work, reported later, we initially used concepts and labels for our management GUI that were the everyday lingo of the security experts who built the GUI. We were very surprised to find that the users who tested the first GUI prototype could not understand what these labels were meant to convey. Finally, as reported

"All staff in the department can write files to laser printer x, Jim the administrator can write files, delete any files from the print queue, pause the printing, and resume the printing at the laser printer x. No one else is allowed access to the printer."

Table 1 – A brief PERMIS policy, in English

earlier, the research by Zurko et.al. found that even their experienced users found some of the concepts and terms used in their GUI to be somewhat confusing. Thus it is clear that in the design and labeling of all security interfaces, great care and early user testing is required in order to bridge the knowledge gap between the security experts and the normal users.

3.1 The design problem

PERMIS access control policies are actually formulated in XML. Originally they were written by hand by the developers and security experts (the power users of [30]). XML policies have several advantages:

(i) They are machine processable and can be easily parsed and understood by the access control decision engine;

(ii) The content of them can be controlled by a DTD or schema thus ensuring that the manager creates syntactically correct policies;

(iii) Open source tools are available for manipulating XML data structures thereby reducing implementation costs.

However, the PERMIS developers rightly anticipated that writing XML policies would not be efficient, intuitive or easy for e-Science node managers. This is inevitable, as machine-readable security policies require an interpretive infrastructure (such as tags to denote different classes of policy items) to avoid ambiguity, which tend to add extra length and verbosity to the resulting policy.

PERMIS XML policies are brief compared with alternatives such as Akenti's[‡] or XACML [18], but still require significantly more verbiage than the same policy written in English. For example, the same policy is written in English in *Table 1* and in XML in *Figure 1*.

In addition, since the tool's intended users – e-Scientists in this case - were predicted to know nothing about PERMIS or RBAC, they would face a significant investment in time and effort to learn the necessary *concepts* before they would be able to write authorization policies. However, e-Scientists want security technology that is like *"a scattering of magic dust"* over their e-Science applications [9]. The current state of the art in RBAC requires significant investment in time and effort to learn its concepts, which is clearly not the desired "magic dust".

It was therefore decided to build a policy editor tool to assist users with RBAC concepts and produce the XML of policies for them, leaving the e-Science node managers to concentrate on the semantics of the policy. Furthermore, it was decided to provide a graphical user interface (GUI), allowing *drag-and-drop* and *point-and-click* operations to further reduce the effort involved in implementing policies.

[‡] See http://www-itg.lbl.gov/security/Akenti/


```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X.509_PMI_RBAC_Policy>
<X.509_PMI_RBAC_Policy
OID="1.2.826.0.1.3344810.6.0.0.0.0.1">

  <!-- only let the people of the department in -->
  <SubjectPolicy>
    <SubjectDomainSpec ID="department">
      <Include LDAPDN="ou=Venables, o=permis, c=gb"/>
    </SubjectDomainSpec>
  </SubjectPolicy>

  <!-- there is only one role: administrator; all others
have default access -->
  <RoleHierarchyPolicy>
    <RoleSpec OID="1.2.826.0.1.3344810.1.1.14"
Type="permisRole">
      <SupRole Value="administrator"/>
    </RoleSpec>
  </RoleHierarchyPolicy>

  <!-- there is only one SOA -->
  <SOAPolicy>
    <SOASpec ID="SOA" LDAPDN="cn=SOA, o=permis, c=gb"/>
  </SOAPolicy>

  <!-- let the SOA assign the administrator role to anyone
in the department -->
  <RoleAssignmentPolicy>
    <RoleAssignment>
      <SubjectDomain ID="department"/>
      <RoleList>
        <Role Type="permisRole" Value="administrator"/>
      </RoleList>
      <Delegate Depth="0"/>
      <SOA ID="SOA"/>
      <Validity/>
    </RoleAssignment>
  </RoleAssignmentPolicy>
 <!-- define the printer domain -->
  <TargetPolicy>
    <TargetDomainSpec ID="printer">
      <Include LDAPDN="cn=printer, ou=Venables, o=permis,
c=gb"/>
    </TargetDomainSpec>
  </TargetPolicy>

  <!-- define the actions -->
  <ActionPolicy>
    <Action Name="print"/>
    <Action Name="delete"/>
    <Action Name="pause"/>
    <Action Name="resume"/>
  </ActionPolicy>

  <!-- define the target access policy -->
  <TargetAccessPolicy>

    <!-- users can only print by default -->
    <TargetAccess>
      <RoleList/>
      <TargetList>
        <Target Actions="print">
          <TargetDomain ID="printer"/>
        </Target>
      </TargetList>
    </TargetAccess>

    <!-- administrator can do anything else -->
    <TargetAccess>
      <RoleList>
        <Role Type="permisRole" Value="administrator"/>
      </RoleList>
      <TargetList>
        <Target Actions="delete pause resume">
          <TargetDomain ID="printer"/>
        </Target>
      </TargetList>
    </TargetAccess>
  </TargetAccessPolicy>
</X.509_PMI_RBAC_Policy>
```

Figure 1 - A brief PERMIS policy, in XML form.

4 Initial prototype

4.1 Design features

The policy editor toolbar is the main focus for interaction (*Figure 2*). Users click on buttons in the toolbar to open windows - forms that are used to fill in parts of the access control policy. There is one button and form for each part of a PERMIS access control policy - and as a starting point the names for these have been taken directly from the technical language of the PERMIS policy elements. The Subject policy window is shown in *Figure 2*.

Each Grid protected by PERMIS will refer to or contain at least one LDAP directory[§], which is a repository for the X.509 attribute certificates used to store users' roles. The policy editor is configured to browse these LDAP directories, and represent them as trees (see the tree marked Directory in the lower quadrant of *Figure 2*) in its domain windows (activated by the Subject, SOA, and Target buttons in the toolbar).

In normal operation the user browses the directory tree until (s)he finds the desired entry (for example *University College London Department of Computer Science*), which can be double clicked to enter it into the LDAP DN field. The user indicates that this DN should be included in the Subject policy by clicking the Include button. The policy writer then types a nickname for the *Subject domain* he is defining in the Domain ID field, for example "UCL", which he will have to refer to in other windows. The information is added to the policy by clicking the OK button, and the window is then closed by clicking the Close button.

[§] the Grid may use an external LDAP directory – e.g. LDAP directory of a collaborating institution. In extreme cases a Grid may not need its own LDAP server, but usually it does.

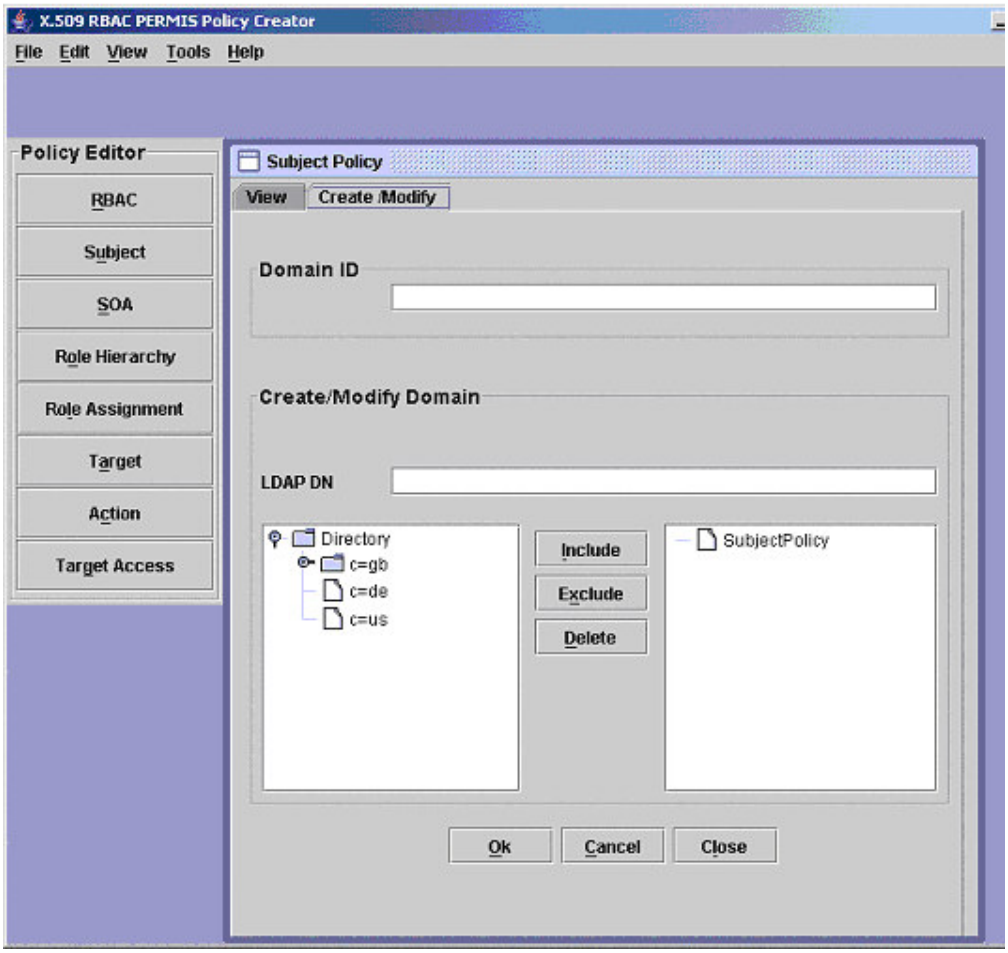

Figure 2 - Example screenshot from the initial prototype, showing the main interface buttons in the policy editor toolbar, and the Subject policy window, closely resembling the windows for SOA, and Target policies. LDAP directories can be expanded and collapsed (lower left-hand corner of window).

In some cases the LDAP directories may not be available for some reason[**], so no LDAP tree will appear in the window. The user will then have to type in DNs into the LDAP DN field from memory instead of seeing, pointing and clicking.

The two most important policy windows are the Target Access policy window and the Role Assignment window (*Figure 3*). These two windows refer to policy items defined in the other windows by using combo-boxes (see SOA, Roles, and to this Subject Domain boxes in *Figure 3*). The combo-boxes contain the ID nicknames users created in previous windows - so for example if the user typed UCL into the Domain ID field in *Figure 2*, it could be selected in the to this Subject

[**] Some of the reasons are: a) security restrictions – LDAP directory may not be publicly browsable, b) firewalls - attempting to connect from a location disallowed by the institutional policy, c) network unavailability – the manager is working on the policy whilst on a plane or train

domain combo-box in *Figure 3*, indicating that the user was referring to the *Subject domain* he defined as University College London's Computer Science Department.

4.2 Efficiency with intermediate to expert users

Informal comparative evaluation by usability researchers and PERMIS developers showed that the first prototype dramatically improved the efficiency with which PERMIS policies could be written. At least one developer reported that policy creation times were shortened by more than 50%. The prototype therefore showed the planned improvement in usability for users who are familiar both with it and PERMIS policies - predicting a considerable increase in productivity for intermediate and expert users.

It was therefore decided to focus further development efforts on increasing the speed with which novice users could progress to intermediate status, by lowering the conceptual barrier that they would have to overcome to use the tool *effectively*.

4.3 Usability trial study

Three researchers from UCL were recruited (2 computer scientists and a biological anthropologist). They were similar to the target user group – e-Scientists – in that they were new to PERMIS and did not have specialist knowledge of computer security, but were numerate and involved in the scientific process.

There is ongoing debate within the HCI community as to appropriate numbers of participants for usability trials, which seeks to minimise costs by recruiting the fewest participants that will give useful results. Proponents of discount usability methods have drawn curves of proportions of usability problems found against numbers of participants per trial. They claim that greatest cost/benefits ratios are obtained with between 3 and 5 participants, that 3 participants will uncover approximately 70% of usability issues, and that additional participants bring diminishing returns [15] [14]. It has been argued that assumptions underlying the curve are optimisitic, do not take into account the variablility between participants and tasks, and that more participants are likely to be required [27]. Some practitioners recommend 4-6 participants when participants are asked to achieve predetermined goals during the trial (i.e. write a policy to do X) rather than achieve their own goals (i.e. write any policy) [e.g. 24], while others only feel comfortable with 8 [cf. 19]. However, the greatest risk of using few participants is of missing important usability problems [11] rather than of giving the problems discovered an importance they do not warrant. Conclusions based on 3 participants (this user trial) and 5 participants (the final prototype's user-trial) entail a larger risk that usability problems discovered are illusory than trials with more participants, and caution should be used in generalising from them to all e-Scientists. The greater risk however, is that these trials will not find all the important usability problems, and that some will remain undetected.

The three participants were given specifications for a simple access control policy and 15 minutes to implement it using the prototype policy editor. Audio was recorded during the session, and video taken using screen recording software.

The test specification could be implemented by the test facilitator in 5 minutes. None of the participants completed the specification after 15 minutes. One participant, a Grid developer, was given extra time and had not finished implementing the specification after 30 minutes when the trial was stopped.

There were several reasons that participants found it difficult to complete the task. We address the most fundamental of these below, i.e. problems relating to users not knowing what needs to be done.

Toolbar and window labels. None of the participants understood the label SOA. Two of the participants did not understand RBAC. One of the participants thought that what should have been entered into the SOA window was a role that should in fact be defined in the Role Hierarchy policy. One of the participants correctly identified the purpose of the Subject and Targets policies, but ascribed the purpose of the Subject policy to the Target policy, and vice versa.

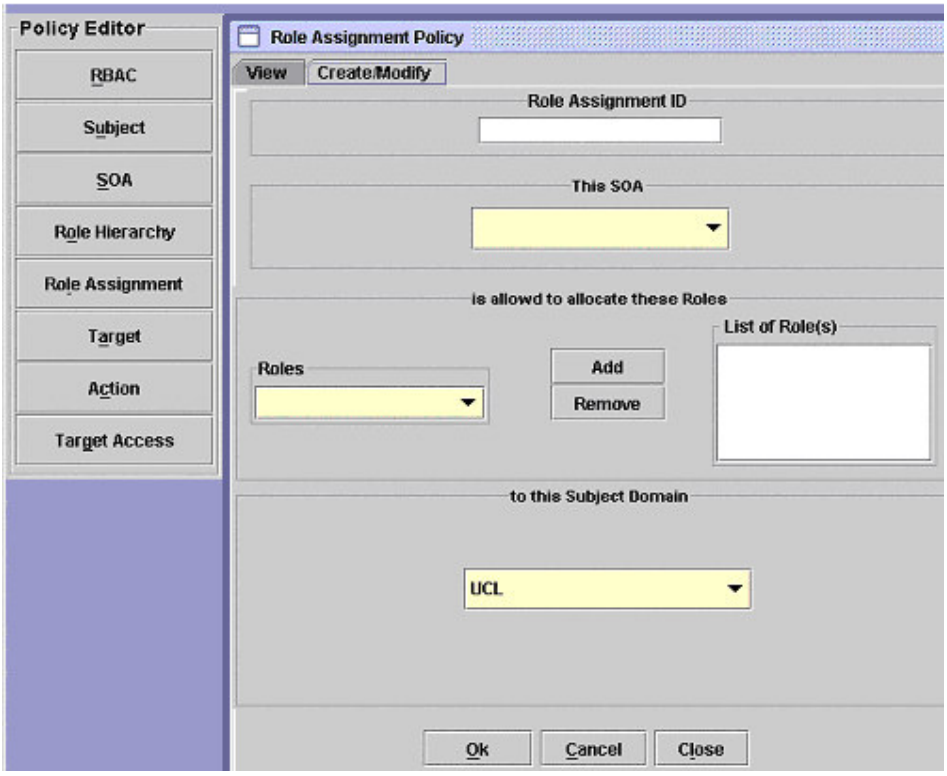

Figure 3 - The Role Assignment policy window (The initial prototype). Combo-boxes contain the IDs created by users in other windows, so that policy items defined in these other windows can be referred to.

These issues point to a fundamental problem. In order to formalise one's access control requirements into a machine enforceable policy, one must understand how the policy space has been structured, and divided into parts. Since the users are not security specialists, they should be guided to consider aspects of access control policy they may not otherwise have thought of. Critically, they must be guided to form policies that are complete with respect to the infrastructure's requirements, so that the resulting policies are enforceable. We label this as the ***policy components problem*** – or as one user said, *"R-what?"*.

We believe that the most probable outcome of the policy components problem will be as we observed in the trial – users being unable to write working policies - their policies are incomplete and so the authorisation infrastructure cannot interpret them sufficiently to enforce them. In the PERMIS infrastructure, having no policy to enforce leads to a fail-safe outcome – all users are denied access. While this is a negative outcome, it is at least easy to detect.

A more pernicious problem would be if users' misunderstanding of policy components caused them to write policies that create significant security holes. For example, policies that give user-level access to the wrong people (potentially jeopardising confidentiality, integrity and availability), or make the wrong people *SOAs*. We believe that the greatest risk of mistaken policy components would be from users misunderstanding the scope of the *Subject* and *SOA* components as being larger than they are meant to be – for example, they may define the Grid users as Salford University rather than Salford University's Information Systems Institute. With these kinds of faulty policies, the vulnerability is that more people have been given the potential to get access to the Grid than is desirable or intended. However, the PERMIS system has some safeguards against these vulnerabilities being exploited by attackers - a mistaken policy is a necessary but not sufficient requirement for attackers to gain access. For example, to have access, as well as being granted the ability to become a Grid user by a faulty policy, attackers would also need to have a digital role AC

generated for them and stored for them in the Grid's LDAP directory; PERMIS splits policy writing from AC generation/distribution with the intention that they are done by separate people - the digital role AC necessary for the attack therefore may not be supplied. From what we have observed we believe there is little risk from these kinds of mistaken policies. In our limited sample, when users' mental models of the policy components were wrong, they were wrong in kind rather than extent. However, the policy components problem remains a source of vulnerability.

Exclude button. The specification required that *"people from Germany be prevented from using a resource",* which would have been the case without any action on the users' part - PERMIS denies all access except that which has been explicitly allowed. The only participant who attempted this part of the specification incorrectly tried to define a *Subject domain* with Germany as an *excluded* node. This mistake showed that the user did not fully understand the function of the Exclude button (see *Figure 2*), since Exclude can only be used after a superior node has been *included*. However, this pointed to a more fundamental misunderstanding. By attempting to exclude something that was mentioned nowhere else in the policy, the user showed they did not understand it was *excluded* by default, and therefore that the user did not understand the underlying *deny all access except* nature of PERMIS policies[††].

This misunderstanding is another fundamental block to creation of access control policies, which we label the ***policy paradigm problem***. This will be a common issue for people who are not intimately familiar with access control lists or similar mechanisms. They will not know what objects they should put into the policy, and what they should leave out. We predict that most novice users would have a mental model of the policy paradigm as similar to *explicitly grant and deny access.* In most cases this would lead to inefficient creation of workable policies – where users unnecessarily explicitly deny access to groups of users who would be denied access simply by not mentioning them in the PERMIS policy. The resulting policy would achieve what the users intended, but be longer than it needed to be.

In rare cases where users have a mistaken *grant all access except* conception of the policy paradigm, we believe it would lead to users writing non-functioning policies, rather than insecure ones: actions intended to deny access to unwanted groups of people (under the assumption that the desired access is granted by default) will fail to grant the desired access in PERMIS. This is another fail-safe condition, which is easily detected by policy writers – no-one would be given access by the resulting policy.

Another fundamental issue is the problem of ***unique naming.*** In contrast to the policy paradigm and components problems, it is not a problem of knowing what needs to be done. Users intuitively understand that unique names must be given in access control policies. It is a problem of knowing how to do what needs to be done. None of the participants were able to write syntactically correct LDAP distinguished names (DNs) to refer to authorisation *Targets*, *Subject* domains or *Sources Of Authority* – all core tasks in writing PERMIS access control policies. Whereas the policy components and paradigm problems would be serious in all predicted uses of the policy editor, the correctly formed LDAP DN problem observed was an artefact due to not yet implemented functionality - point and click DN entry from a browsable directory (see *Figure 2*). This problem would not usually occur when the missing functionality is present, as will be demonstrated in the final prototype's user-trial. Only in unusual circumstances would a Grid's LDAP directory not be browsable. However, users' inability to manually enter DNs is a cause of concern should this circumstance arise.

[††] The observed use of the Exclude button on Germany would have been correct if a superior node had first been *included*. However, a superior node was available ("Directory" –see *Figure 2*) and the user did not seek to *include* it nor create an alternative superior node such as "the world" to *include*.

5 The final prototype

Several rounds of evaluation and improvements were made during the policy editor's iterative development. A summary of the findings, and resulting changes to the design of the GUI, is presented in this section.

5.1 Design features

The final prototype retains the overall look and feel of the initial prototype. However, to tackle the ***policy components problem***, an effort was made to make the language in its interface more intuitive for people with little background in RBAC and PERMIS. Steps were also taken to tackle the ***policy paradigm problem***. Work on the language in the policy editor's UI will be described first.

5.1.1 Promoting understanding of policy components

A user's initial interaction with the policy components is through the labels in the policy editor's UI, supplemented by bubble help. In the first prototype, these were from the technical language of role-based access control. Specialist technical language - jargon - serves an important purpose for experts, offering shorthand for well-understood concepts and signalling membership of a group of whose members share a certain level of understanding. However, non-expert users naturally refer to the everyday meaning of terms that serve as shorthand to experts. The resulting misunderstandings are a source of error and frustration. Explaining the meaning of terms is not a solution: non-experts do not understand the underlying concepts in sufficient depth, and with infrequent use, the everyday meaning of a term will always be the first one that comes to mind. Jargon will always be a source of error and frustration for non-experts. Computer security has produced many examples of labellings that prove difficult and misleading for non-expert users. Encryption, which uses terms such as "key", "public", and "private" in a different way than suggested by everyday language, is a primary example [26]. Rather than making users acquire the extensive knowledge that would enable them to use jargon – i.e. turning them into experts, usability techniques can be applied to re-label concepts in a way that suggests correct actions.

Conceptual design (CD) is a well-known HCI technique that taps into the language and models of users [17]. Many users require labels that imply concepts that are already known to them, using words from their everyday language with meanings that are analogous to the technical concepts. New users are unlikely to have natural language that **exactly** matches the tightly defined technical concepts of access control – there are likely to be areas where the existing concepts do not imply enough of the technical concepts to be useful, and even areas where natural language extends beyond the technical concepts in misleading and unhelpful ways. For example, users were confused by the public and private "keys" in PGP that do not work like keys as the users knew them, and this was one of the reasons users were so unsuccessful in using PGP to send encrypted email [cf. 26]. We therefore used Conceptual Design to select labels from users natural language, to make the policy components intuitive to e-Scientists - our target user group.

The *metaphor evaluation matrix* [cf. 1] is one of the key methods in Conceptual Design. The matrix (*Table 2*) was applied to candidate labels (including those used in the first prototype). This qualitative technique requires the analyst to fill cells in the matrix for a label or metaphor through a process of guided introspection, taking the user's point of view [5]. Empirical techniques (interviews, questionnaires, etc.) can be used to supplement introspection if required.

Labels such as Users, Managers, Resources, and Access Control give large improvements in the amount of supported functionality that they imply (i.e. M+P+ cell entries) compared to the technical language of the initial prototype. For example, the initial prototype's Subject label would imply specification of something to which the policy applied, but would not be enough for a non-specialist user to understand what kind of thing should be specified. However, a candidate alternative for this label - Users - would imply the specification of persons who wish to use the protected resource - almost a perfect match for the functionality of this policy component.

However, the analysis suggested that some of the candidate labels contained *conceptual baggage* that would cause misunderstandings with some users. For example: Users implies the specification of individual persons who wish to use a protected resource - something which cannot be specified in PERMIS policies, because PERMIS policies give access to *roles* rather than individuals.

A questionnaire was designed to test the intuitiveness of two candidate versions of labels – label versions 2 and 3. Version 2 labels were our team's intuitive attempt to create better labels, as a result of the initial user trial. Version 3 labels were the result of applying the metaphor evaluation matrix, using version 2 labels as a starting point. Each questionnaire was in two sections. The first section gave the labels and a policy goal, and asked respondents to describe and give an example of what should go in the relevant window. This gave a sense of respondents understanding of PERMIS functionality from knowing the labels alone. The second section of the questionnaire gave each label a different policy goal, and policy statements that had been derived from the goal - one per label. Respondents were asked to match labels to policy statements by drawing a line between them. This measurement is similar to the preceding one but gives respondents extra cues that are like those that would be available when looking at the actual prototype.

A call to participate was sent out by email to UCL's and the University of Cambridge's internal mailing lists for people who were interested in e-Science (predominantly e-Scientists, system administrators and e-Science researchers/developers)– approximately 207 people. In addition, a call to participate was sent out to the TrustCoM project (80 people), and 3 people who had downloaded PERMIS and had previously answered a requirements questionnaire. No material inducements were given to participate (entry into a prize draw, etc.). 8 responses were received, giving a response rate of 2.8%. This result will be used in the Discussion section as an index of our user group's interest in security.

The scores for each questionnaire section were aggregated. Version 2 labels scored an average of 9.25 out of 16 (57.8%; minimum score = 6.5, maximum =10.5), and version 3 labels scored an average of 13.9 out of 16 (86.7%; minimum score =12, maximum =16). This difference was statistically significant (paired t=6.28, df=7, p=.000 two tailed), showing that version 3 labels are more intuitive than version 2 labels, with a large effect size (f=0.78).

The improvements in intuitiveness scores of individual labels varied greatly between versions two and three (Table 3). User privileges, Account administrator privileges, Where users are from and Resource functionality showed particularly good improvements. The relatively low score for Resource functionality in particular was a matter of concern, leading to a reassessment and production of a revised set of labels (see *Figure 4*).

5.1.2 Tackling the policy paradigm problem

We pursued two techniques against the policy paradigm problem: instructional text in the GUI, and UI behaviour. Both these were supplemented by work we had done with labels for the policy components problem.

Instructional text was added to a prominent part of the GUI to improve users' understanding of the nature of PERMIS' policy paradigm (see red text at bottom of *Figure 6*). This text was placed in the first window that we expected users to interact with, based on our observations during the initial prototype's user trials. All of the users from these trials had opened this window as the first thing they'd done. However, we could not guarantee that all future users would do so, nor that opening the window guaranteed that the users would read the text. We therefore supplemented this with GUI behaviour in the windows where users where at risk of the policy paradigm problem.

Figure 5 shows the UI elements common to all windows where users could fall foul of the problem. Our goal was to encourage novice users to grant access by selecting DNs and clicking the Include button. This built upon the labels chosen to combat the policy components problem, for example: Where users are from. This label implies the selection of DNs that would be granted access, further implying that they do not already have access, leading to the conclusion that PERMIS *denies all access except* (contrast this with a hypothetical label Where users are not from).

	M+	M-
P+	Desirable Features provided by the policy component and supported by the metaphor. Leads to correct use of system.	Undesirable Features provided by the policy component but not supported by the metaphor. Leads to underused features.
P-	Very undesirable Features implied by the metaphor but not supported by the policy component: *Conceptual baggage* Leads to user errors.	Not important Features not implied by the metaphor and not supported by the policy component

Table 2 - A metaphor evaluation matrix [cf. 1]. M+, metaphor implies, M-, metaphor does not imply; P+, policy component supports, P-, policy component does not support.

Version 3 Labels	% correct	Version 2 Labels	% correct
User privileges	84.4	Access control	21.9
Account administrator privileges	87.5	Role allocations	37.5
Account administrators	87.5	Managers	81.3
User types	84.4	Roles	56.3
Where users are from	100.0	Users	43.8
My protected resources	90.6	Resources	90.6
Resource functionality	59.4	Actions	37.5
Policy object ID	100.0	Policy number	93.8

Table 3 - Intuitiveness of version 2 and 3 labels. Percentage correct is the proportion of correct questionnaire responses for each label. 8 respondents answered two questions about each label.

This clear message is complicated by the presence of an Exclude button, which implies a *grant and deny access* paradigm. However, the Exclude button is useful; it represents advanced functionality for intermediate to expert users. It allows users to speed up policy creation where part of an organisation should be granted access and other parts should be denied it. In a simple example, using the Exclude button could do this in two steps, by granting access to the University of Salford and removing access from its Maths department. Without the Exclude button, each department would have to be individually granted access. This would require as many steps as departments and be a much more laborious task.

We therefore wanted to prevent novice users from misusing the Exclude button - explicitly using it on all the users they did not want to grant access. This erroneous behaviour would not compromise the intended effect of the policy, but would be grossly inefficient, perhaps causing users to lose faith in the software. The same policy outcome could be achieved simply by not mentioning these users in the policy - PERMIS would deny them access by default.

To prevent incorrect use of the Exclude button, it is greyed out until a DN has been *included*. Moreover, users are forced to employ the Exclude button correctly: for example, the Maths department can only be *excluded* if the University of Salford is already *included* and also selected –

Figure 4 – Toolbar from the final prototype, showing the revised labels

binding the two DNs in a hierarchical pair. Any other use of the Exclude button results in an error message that instructs users how to employ it correctly.

5.2 Usability trial study

Five numerate postgraduates went through the trial, having 40 minutes to implement two access control policies to our specifications – one exercising basic functions of the policy editor, and the other some of its more advanced features. Both the test specifications could be completed by the test facilitator in a total of 7 minutes.

Two users successfully implemented the whole of specification one, with the other three users completing 85%, 69%, and 46% of its tasks (the user scoring 46% abandoned the trial early), giving an aggregate task completion rate of 80% for basic tasks in policy creation. Four users went on to the second specification. Two of these users successfully completed the whole of specification two, with the other two users completing 67% and 33% of its sub tasks within the available time. This gives an aggregate task completion rate of 75% for advanced tasks (excluding the user who did not attempt them).

This usability trial showed that changes made to the UI had resulted in meaningful improvements in areas where there had been problems observed in the earlier user trial:

Policy components problem. Users completed tasks successfully requiring the understanding of all of the policy components that caused problems in the first user trial. This showed that the labelling was much more intuitive than in the initial prototype. The task completion rates for basic policy tasks varied for each toolbar label, and are shown in *Table* .

Though much more successful than the initial prototype, a small systematic problem was observed with the final labels. A small proportion of users attempted to enter themselves into the policy as having the power to write policy, showing a fundamental misunderstanding of the *Account Administrators* policy component and PERMIS architecture more generally. It is not clear if this ***problem of user self-reference*** is caused by conceptual baggage with the label Account Administrators, or something else, for example: a deeper clash between the users' underlying model of authorisation and the PERMIS architecture. If it is the former, then it may be avoided by

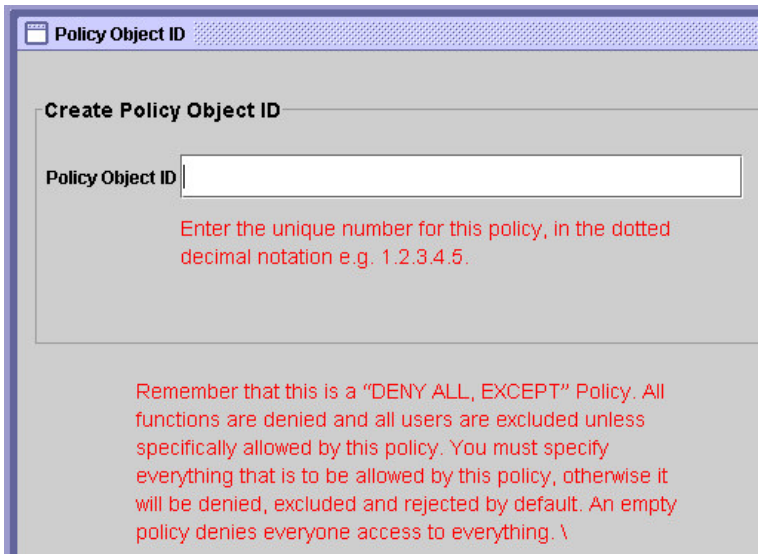

Figure 6 - Explanatory text in the UI laying out the underlying nature of PERMIS policies. (Prototype 2)

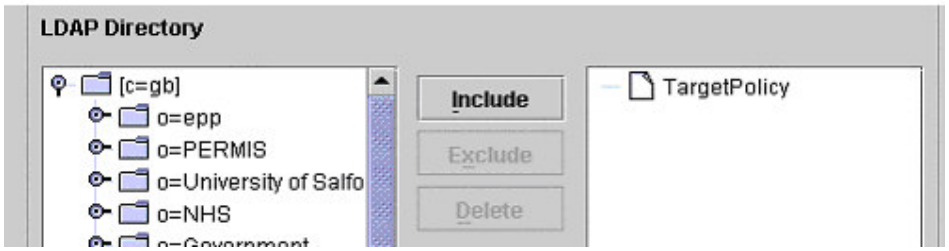

Figure 5 - UI behaviour tackling the policy paradigm problem

choosing a different label. If it is the latter, then explicit education may be required such as tutorials (which many users avoid), or even architectural change.

Policy paradigm problem. A task was built into both user trials to test whether users understood the *deny all access except* nature of PERMIS policies, which had been an area of concern for the design team. If users did understand, they would avoid using the Exclude button in this task, whereas if the UI design caused them to misunderstand, users would mistakenly use the Exclude button in the task. In the initial prototype's user trial, only one user got far enough to attempt this task, and failed it. Changes were made to labelling and UI behaviour to make the *deny all access except* nature of PERMIS more clear, and in the final prototype's user trial four out of five users passed the task.

Unique names problem. In the initial prototype's user trial, none of the participants were able to write correct distinguished names (DNs) to refer to authorisation *Target* and *Subject* domains or *Sources Of Authority*. The final prototype implemented point-and-click selection of these policy items.

Table shows that users of the final prototype were much more successful at entering correct DNs. Only one of the users in this trial already had experience of LDAP and DNs, however, the average task completion rate for DN tasks with the final prototype was four out of five users successful. Its point-and-click UI for LDAP DNs greatly contributes to usability in the PERMIS policy editor.

Policy component	No. users completing task (out of 5).	
Policy Object ID	4	(80%)
Where Users Are From	4.5*	(90%)
Account Administrators	3.5*	(70%)
Users Roles	5*	(100%)
Account Administrators Privileges	4*	(80%)
My Protected Resources	4	(80%)
Resources Functions	4	(80%)
Users Privileges	2	(40%)

Table 4 - Task completion for each policy component in the final prototype, specification 1.
*Average of two tasks

6 Discussion

We set out to improve the usability of the PERMIS system for writing authorisation policies for e-Scientists. This was achieved by employing 4 techniques:

Allocation of function - the task of writing XML was reallocated from the user to software, thus producing a large increase in the efficiency with which intermediate and expert users could create policies.

Usability trials - identified particular features of the UI that could be altered to improve the speed which novice users could appreciate the functions of different parts of the software.

Conceptual design - generated design ideas and informed design decisions for improving learnability for novice users, and so lowering the costs of adopting the software.

Low fidelity prototyping/questionnaire studies - were used to test design ideas without using precious programming resources.

Along the way, the project team identified and addressed three generic problems in making authorisation policy creation tools usable:

(i) Policy components problem.

(ii) Policy paradigm problem.

(iii) Unique names problem.

Our user trials show that we have met with some success. PERMIS has a sophisticated and therefore complex model of authorisation, and yet users were able to implement much of two authorisation policies with no training or access to documentation, and in a short time. We feel that the approach we have taken is necessary, has yielded a better tool, and a better understanding of what users need in order to write authorisation policies.

However, we believe that even better usability can be achieved by carrying on this work with the techniques we have used. There is a tension between designing a simple system image for novice users, and the complexity added to this image by advanced features for intermediate and expert users. This was evident in our experience of the ***policy paradigm problem:*** the Exclude button was an advanced feature that contravened the PERMIS *deny all access except* paradigm. It confused some novice users, causing them to write a policy less efficiently than they might, and frustrated them by receiving unexpected error messages. Further gains in usability may be made by exploring alternate labels for these advanced features. The standard HCI technique of hiding advanced features (under Advanced buttons, for example) may also be useful.

Our test bed held a small namespace of individuals, organisations and resources. When deployed in the real world, authorisation systems' naming UI will face a much more difficult test. Users will have to contend with problems such as several people in the same directory having the same first name and surname [8]. This may be complicated by the technology used in the directory, for example LDAP. Despite being pervasive, LDAP has several open issues which it may not be possible to overcome using the usability techniques we have employed, for example: common components of LDAP DN's (such as Organisational Unit or Locality) are ambiguous in many circumstances [8], and so may increase the difficulty of locating an individual rather than making it easier.

We believe that our efforts have led to a tool that will allow novice users to create safe policies - they will either work as the users intended or deny access to everyone. However, more work is required to validate this. In addition, our belief is based on novice users implementing policies to *our* specifications. We have not yet studied novice users generating policy goals for themselves. Do novice users require support in formulating policy goals to meet their own and their organisations' needs? What knowledge and experience is required to do this? What tools could be built to supply them?

We have found that once the users' fundamental (policy components and paradigm) problems of what-needs-to-be-done to write access control policy have been identified and addressed, thorny problems still remain - users have to work out how to do what they now know needs doing. Small aspects of UI design can help or hinder this stage of policy writing, and we would have liked to have been able to employ more of the conventional HCI techniques to make this even easier.

The project team noted a number of points during this development that other e-Science security projects may wish to consider. Incorporating established usability techniques improved our development process and our resulting software (see above), and having usability specialists on the team made this easier. In particular, we found that testing often and early was of benefit, as it gave us powerful insights into design when we had resources to do something about them. The availability of usability data from testing shortened the debate about design decisions. In contrast, design discussions could become quite drawn-out where data was not available. This was particularly an issue with the analytic (rather than empirical) usability technique *heuristic evaluation* [12] [13], which we later found to have made valid predictions.

We were surprised by the difficulty of recruiting test participants from among the e-Science community - our calls for participation had a 2.8% response rate for questionnaire studies (see section 5.1.1), and a 0% response rate for usability trials. User trials could only be conducted by using face-to-face meetings for recruiting. As a result, we were not able to achieve the level of testing or the confidence in our results we desired, and this made our development process more difficult. PERMIS and e-Science are relatively new - there were few members of an existing user group that would be motivated by self-interest (in having improved software) to take part in testing - participation in usability trials can take up to an hour. We were forced to create low impact test instruments to make participation more attractive - short questionnaires that implemented some aspects of low fidelity prototype testing. However, we found these to be an excellent way of collecting data - quick, cheap, informative, and requiring no programming resource. Having other means of attracting test participants would have been beneficial - budgeting to pay for test participants, or having a large group of project partners who could be put under social or contractual pressure to participate. Future projects may make more extensive use of usability inspection methods - such as *heuristic evaluation* and *cognitive walk-through* or its more streamlined derivatives [25]- which do not require any test participants, but can identify significant numbers of issues revealed by usability trials [7].

Though creating difficulties for our testing activities, the low response rate has validated our approach and focus on usability to create an easy access, lightweight authoring tool. Participating in our studies would have taken our users relatively little effort, yet their enthusiasm for participating was significant in its absence. A similar, if not greater, amount of effort would have been required to learn the original policy components and UI before the HCI makeover, but this is an effort that these users are clearly not willing to make (unless they have to).

Whilst some usability interventions (such as *reallocation of function* of XML creation from the user to the computer) required deep changes to our software - other improvements were almost cosmetic in terms of the coding required to implement them (changes to button labels lead by *conceptual design*). These label changes nevertheless brought significant improvements to usability. Good software engineering practices made these improvements easier to implement - however, such practices have much wider benefits, for example making software localisation more efficient.

The utility of our questionnaire studies showed that it was not necessary to have working code to get useful design data. We conclude that in some instances it can be preferable to test in the absence of a working prototype, as design ideas can be tested without throwing code away [24]. This leads to the conclusion that testing and freezing of UI designs should be done as much as possible *before* coding begins, or if not then as early as possible - giving the developers greater flexibility in design, and a more efficient and often more rewarding software engineering process [6].

There is a perception that the difficulty in getting e-Science technology to work drives security considerations to the back of the queue, and that security mechanisms impede application users rather than support them. This perception causes users to find ways of subverting the security mechanism [22]. We feel that incorporating usability principles and techniques into the development of e-Science security software can make it more attractive to the wider e-Science community, by lowering the entry barrier, increasing its uptake and appropriate use.

References

1. Anderson, B., Smyth, M., Knott, R.P., Bergan, M., Bergan, J. and Alty, J.L., Minimising conceptual baggage: making choices about metaphor. in *People and computers IX*, (Glasgow, 1994), Cambridge University Press, 179-194.
2. Balfanz, D., Durfee, G., Smetters, D.K., Grinter, R.E. "In Search of Usable Security: Five Lessons from the Field". IEEE Security and Privacy, Sept/Oct 2004, p19-24.
3. Britton, D., Clarke, P., Coles, J., Colling, D., Doyle, A., Fisher, S.M., Irving, A.C., Jensen, J., McNab, A. and Newbold, D., A Grid for Particle Physics - from testbed to production. in *UK e-Science All Hands Conference*, (Nottingham, 2004).
4. Chadwick, D.W., Otenko, A. and Ball, E. Implementing Role Based Access Controls Using X.509 Attribute Certificates. *IEEE Internet Computing*. 62-69.
5. Clark, L. and Sasse, M.A., Conceptual Design Reconsidered - The Case of the Internet Session Directory Tool. in *HCI '97*, (Bristol, 1997), Springer, 67-84.
6. Cooper, A. The Inmates Are Running the Asylum: Why High-tech Products Drive Us Crazy and How to Restore the Sanity. Sams, 1999.
7. Desurvire, H.W. Faster, cheaper!! Are usability inspection methods as effective as empirical testing? in Nielsen, J. and Mack, R.L. eds. *Usability inspection methods*, John Wiley & Sons, Inc, New York, 1994, 173-202.
8. Gutmann, P. Godzilla crypto tutorial, Part 2, Peter Guttmann, 2002.
9. Humphrey, M., Grid Security: Are we making progress? Presented at the 2004 *UK Workshop on Grid Security Experiences*, (Oxford, 2004).
10. ISO 9594-8/ITU-T Rec. X.509 (2000) The Directory: Public-key and attribute certificate frameworks
11. Law, E.L.-C. and Hvannberg, E.T., Analysis of Combinatorial User Effect in International Usability Tests. in *CHI2004*, (Vienna, Austria, 2004), ACM, 9-16.
12. Nielsen, J. Heuristic Evaluation. in Nielsen, J. and Mack, R.L. eds. *Usability inspection methods*, John Wiley & Sons, New York, 1994, 25-62.
13. Nielsen, J. How to Conduct a Heuristic Evaluation, 1994.
14. Nielsen, J. Why You Only Need to Test With 5 Users, useit.com, 2000.
15. Nielsen, J. and Landauer, T.K., A Mathematical Model of the Finding of Usability Problems. in *INTERCHI '93*, (1993), ACM, 206-213.
16. NIST RBAC Standard
17. Norman, D. Some Observations on Mental Models. in Gentner, D.A. and Stevens, A.A. eds. *Mental models*, Erlbaum, Hillsdale, NJ, 1986.
18. OASIS. OASIS eXtensible Access Control Markup Language (XACML) v1.0, 2002.
19. Rubin, J. *Handbook of Usability Testing*. John Wiley & Sons, Inc, New York, 1994.
20. Saltzer, J.H., and Schroeder, M.D. "The Protection of Information in Computer Systems," Proceedings of IEEE, 63(9), 1278-1308, 1975.
21. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E. "Role Based Access Control Models". IEEE Computer 29, 2 (Feb 1996), p38-43.
22. Sasse, A., Brostoff, S. and Weirich, D. Transforming the 'weakest link' — a human-computer interaction approach to usable and effective security. *BT technology journal*, *19* (3). 122-131.

23. Schaad A., J. Moffett, and J. Jacob, "The access control system of a European bank - a case study." presented at Sixth ACM Symposium on Access Control Models and Technologies (SACMAT), Chantilly, VA, USA, 2001.

24. Snyder, C. Paper prototyping: The fast and easy way to design and refine user interfaces. Morgan Kaufmann, London, 2003.

25. Spencer, R., The Streamlined Cognitive Walkthrough Method, Working Around Social Constraints Encountered in a Software Development Company. in *CHI 2000*, (The Hague, The Netherlands, 2000), ACM Press, 353 - 359.

26. Whitten, A. and Tygar, J.D., Why Johnny can't encrypt: a usability evaluation of PGP 5.0. in *9th USENIX security symposiom*, (Washington, 1999).

27. Woolrych, A. and Cockton, G., Why and When Five Test Users aren't Enough. in *IHM-HCI 2001*, (Toulouse, 2001), Cépadèus Éditions, 105-108.

28. Yee, K.-P., "User Interaction Design for Secure Systems," *Proc. 4th Int'l Conf. Information and Communications Security*, R. Deng et al., eds., LNCS 2513, Springer, 2002, pp. 278–290; http://zesty.ca/sid.

29. Yee, K.-P., "Aligning Security and Usability". IEEE Security and Privacy, Sept/Oct 2004, p48-55.

30. Zurko, M.E., Simon, R. and Sanfilippo, T., A User-Centered, Modular Authorization Service Built on an RBAC Foundation. in *IEEE Symposium on Security and Privacy*, (1999), IEEE, 57-71.

