Access Control for Shared Resources

Erik Wilde and Nick Nabholz
Computer Engineering and Networks Laboratory (TIK)
Swiss Federal Institute of Technology (ETH Zürich)

Abstract

Access control for shared resources is a complex and challenging task, in particular if the access control policy should be able to cope with different kind of sharing and collaboration. The reason for this is that traditional access control system often depend on administrators to set up the foundations of the access control mechanism, in most cases users and their group memberships. The access control model presented in this paper approaches this problem by supporting two different kinds of groups, named groups and resource-based groups. Using the implementation of this model in our application allows to to support a wide variety of sharing and collaboration types between the application's users.

1 Introduction

Many applications require authentication and authorization for managing access to shared data and services (called resources from now on). In many of these cases, it is also desirable to be able to group individuals, and extend authorization in a way that groups can be granted access to resources. This way, groups can be made the foundation of authorization, and while group populations may vary (with individuals joining or leaving the group), the authorization settings of resources do not have to be changed.

The functionality of managing users and groups always has a common set of requirements. However, individual applications often have different resources which should be access controlled, and different access rights which should be managed. This paper describes a generic architecture and implementation for managing users, groups, and access controlled resources.

In the resource control and group management model presented here, groups are treated as resources, i.e. they need authorization for determining who is permitted to add and remove group members. Even the system is modeled as a resource, which results in a clean and consistent design, which can be easily extended to meet new requirements.

2 Application Scenarios

The model presented here can be used in a variety of settings, but has been developed in the context of the *ShaRef (Shared References)* project [5], which is concentrating on managing bibliographic metadata in a collaborative context, and thus requires resource control and group management. The resources in this application are so-called *bibliographies*, collections of bibliographic references, which should be usable by individual users, but also by user groups (e.g., a research group managing a list of related work in their area of research).

Figure 1 shows ShaRef's data model, and it can be seen that apart from the user and group concepts, the most important concepts are bibliographies (as described above), *databases* (collections of bibliographies which share a common set of user and group information), and *workspaces* (which serve as a kind of "shopping cart" in the model, being the interface between user interactions and the persistent storage in the database's bibliographies). Bibliographies and Workspaces contain *references* and *shadows*, the latter being a special kind of reference.

In the context of this paper, the important observation is that access control in our system is based on the level of bibliographies and workspaces (database being the bigger contexts). Thus, the shared resources which have to be managed by an access control system are bibliographies and workspaces, while the contents of these resources can be freely used once authorization has been granted. This scalability in terms of access control make it possible to adapt the system to specific application requirements, for example for determining the scope of payed access to resources in e-commerce applications.

Generally, the application area of the model presented in this paper is any application requiring shared access to resources, and where access should be controlled through user identities and the users' group memberships. The granularity of the resources and the actual rights being managed can be adapted through the generic model presented in Section 3. Furthermore, identification and authentication can be handled internally or externally, so that our system can

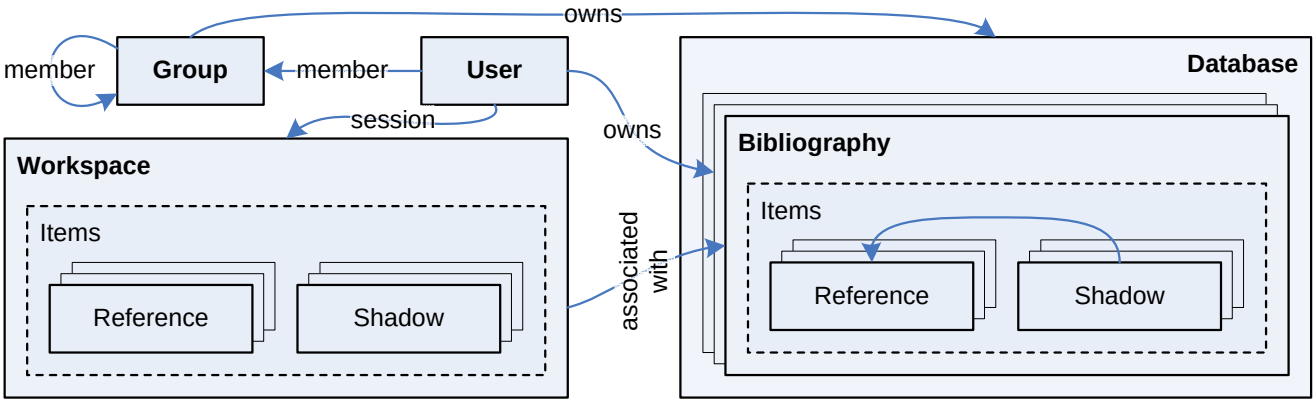

Figure 1. ShaRef Data Model

be combined with existing authentication services, for example for handling customers which are already managed in a separate database.

3 Access Control Model

One of the key design goals was to keep the model as simple as possible, so that it is easy to understand and to apply by users, which typically are not experts in using a collaborative system. The majority of our users are casual users, and the model must be simple enough for them to be easily understandable.

3.1 Users, Groups, and Resources

In our model, *users* and/or *groups* can be members of groups (cyclic structures are prohibited). The idea is that hierarchical structures of organizations can thus be easily reflected in the organization of the users and groups. Determining whether a user is authorized for some action is thus equivalent to testing whether this user is directly or indirectly (through group membership) authorized for this action.

The core concept for authorization is that of *roles*, a role is defined for each resource class, and each role can be played by any number of users and/or groups. A special case is the *administrator* role, which must have at least one user and/or group. Having a role is equivalent to having a certain authorization, and the actions authorized are specific to the concrete application and resource. The *administrator* as a pre-defined role exists for every resource, and being an *administrator* of a resource is equivalent to having the right to add and remove users and/or groups to this resource's roles.

Figure 2 shows the overall model. On the top is the abstract *managed resource class*, defining the *resource administrator* role, which is inherited by all classes. Derived from the abstract base class (arrows with straight lines) are the *system class* (defining system administrators and resource creation rights), the *group class* (a class required for all scenarios), and the *bibliography class* (which is specific for the ShaRef scenario).

The system class is required for system adminstration and resource creation management. Only system administrators are allowed to create users, they manage the user objects in the sense that each user may or may not have the right to create resources, and these creation rights can be changed by system administrators only. Resource creation is managed through roles, too, but these roles have built-in semantics (resource creation, depicted by dashed arrows), and for each resource class, there is one such role in the system class.

The group class defines one additional list (in addition to the *group administrator list* inherited from the managed resource class) for a group's members.

There is the possibility to use hierarchic roles (thin arrows), which means that the individual roles are not independent. Each role can be hierarchic (being a descendant of another role of the same resource class) or independent, with the following reasoning behind these two configurations:

- *Hierarchic Roles:* In case of hierarchic roles, all entries having a certain role are considered having the dependent role, too. For example, in ShaRef, a writer of a bibliography implicitly also is a reader (and in fact, an administrator implicitly is a writer and thus a reader, too). So this means for ShaRef bibliographies, all three roles are structured hierarchically (shown by the thin arrows in the bibliography class in Figure 2).

- *Independent Roles:* If roles are not considered subsets, roles are independent, which means that only entries

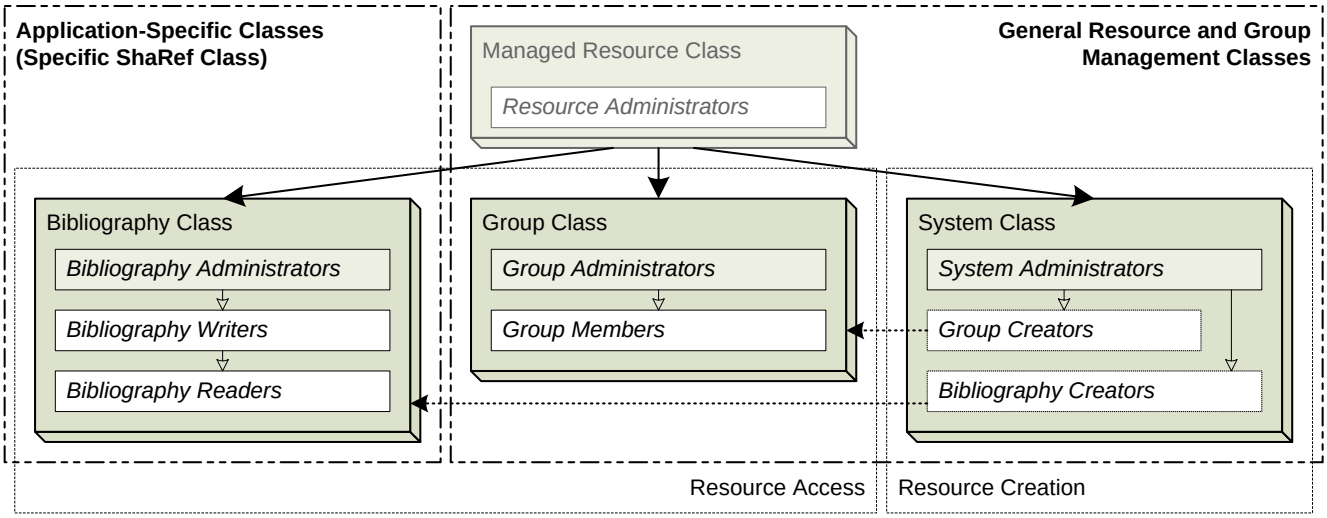

Figure 2. Resource Control and Group Management Model

having the role explicitly are considered to have the associated access right; there is no dependency between an independent role and other role. For example, in ShaRef, system administrators may create groups and bibliographies, but the group and bibliography creation rights are independent (shown by the thin arrows in the system class in Figure 2).

In a given application scenario, developers have to decide which types of resources should be access controlled. For each of these types, a resource class has to be created (as in the box labeled "Application-Specific Classes" in Figure 2), and for each access right, a role has to be defined (either hierarchic or independent). For each of these classes, the system class will have a corresponding *creator* role, which controls the creation of resources of this type.

3.2 Resource-based Groups

So far, roles can be assigned to users and groups, and groups have to be created explicitly (and be given a name, which is why they are called *named groups*). This kind of group management works well for managing group structures which are aligned with real-world organizational structures, such as research groups or departments. For a more dynamic way of managing resources, however, we provide an alternative concept for defining groups, which is called *resource-based groups*.

While a named group is defined by an instance of the group class (by all users and/or groups having the administrator or member role for this group), a resource-based group is implicitly defined by any other resource. When using a resource-based group, it is referenced through a resource and a set of roles, and the members of this resource-based group are all users and/or groups having the specified role(s) on the specified resource. For example, a user may create a new resource and specify that the new resource can be read by all readers of another resource, and if the other resource's readers are being changed, this change is reflected on the newly created resource, too.

Named groups are better suited to support organizational structures, where groups are well-defined and change infrequently. Resource-based groups provide a more flexible and dynamic way of managing resource access, with the risk of unintentional changes being triggered by changes of the referenced resource (this risk is increased by chains of resource-based groups).[1] It is at each user's discretion to decide which group concept is considered more appropriate for a given scenario.

4 Implementation

The first step necessary for using the access control system is successful authentication, which means that a user has to be identified, and his identity has to be verified. After successful authentication, the resource control service provides all the information about permissions of individual users on resources, as well as the required management functionality for this environment. The actual implementation of access control (i.e., the application-level semantics of roles) is application-specific and must therefore be implemented within the application.

[1] To eliminate the risk of "orphaned resources", resource-based groups may not be used within the resource administrator role.

4.1 Authentication

Authentication can be done by either using a built-in database of users and credentials, or by accessing an external service for user authentication.

Using the built-in database makes it easy to use and deploy the service within standalone applications. There is an API for registering new users, so applications can implement their own functionality for managing users and can use this API for managing their user base.

Using an external authentication service is preferable when there is an existing database of users and credentials that should be reused by an application. This is particularly important for implementing *Single Sign-On (SSO)* applications, where users have one centrally maintained identity which they can use to access multiple applications.

4.2 Resource Control Service

Any application that wants to use this service must register newly created resources for access control. The creator of this new resource gets the administrator role. This means that this user has the permission to grant any role for this resource to any other user or group.

It is on behalf of the application developers to define additional roles for their resource types, and to specify what actions are allowed or forbidden for each of these roles. The resource control service answers the question "what are the roles of this user for this resource?". The enforcement of the request's result is application-specific and has to be implemented inside the application.

4.3 Clients

The resource control service provides the necessary functionality to implement a user or application interface for access control and management. Application developers access the resource control service via Java RMI, which makes is possible to deploy distributed configurations, where the clients are distributed throughout a network, while the access control service runs as a centralized service..

As an existing client implementation, we provide a Eclipse plugin for the management of groups and users. The plugin is a user interface for the resource control service. It displays information about users, groups and application-specific resources. A user of this plugin can set the required roles for his resources. He can assign a role to any users and/or groups within the system. If the user has the permission to create groups or other resources, this task is also supported.

5 Discussion

Our access control model is a *Discretionary Access Control (DAC)* model. Individual users may grant and revoke privileges on their resources, which is pure DAC, but our model of groups and the system as resources as well is a particularly powerful variation of the general DAC model. This model allows a user to setup collaboration easily.

5.1 Related Work

The access control model presented here is based on existing access control models. They are well known and described by FERRAIOLO et al. [2], the two most important models are the following:

- *Discretionary Access Control (DAC)* permits granting and revoking of access privileges at the discretion of the individual user. The user may grant privileges for the resources under his control to other users without intervention of a system administrator.

- *Role Based Access Control (RBAC)* bases access control on the function a user has in an organization. A role can be thought of as a set of permissions within the context of a organization. A user can not pass access permissions to other users at his discretion.

Roles are a very powerful concept described in detail by EDWARDS [1]. However, collaborative work (as in our application scenario) often takes place in groups which are not reflected in the organization's structure. Therefore, RBAC often has difficulties to support the requirements of collaboration if the collaboration and sharing of resources is not tied to very strict rules and workflows.

The idea of distributed administration of access rights, which is behind DAC models, supports the requirements of collaboration much better. Many shortcomings of the DAC model which are mentioned in the description of the access matrix model (another name for DAC) in TOLONE et al. [4] can be avoided by our model by the properties described in the following section.

5.2 Contributions

While the basic system model is that of a DAC model, there are some special facets of our model which are not found in other DAC systems. In particular, the model has been designed to be as consistent as possible, which is reflected in the fact that groups and the system are modeled as resources as well.

Groups are modeled as resources (a built-in resource class). A group is controlled by normal users having the

administrator role for the group. More traditional DAC systems treat groups as a totally different kind of resource, and this more limited model can be easily mapped to our model by only using groups where the system administrators are also group administrators.

The system is modeled as a resource (a built-in resource class), and there is exactly one instance of this class at any given time. The administrators of the system resource controls all resources. System administrators may create any resource and remove any resource from the system. But a system administrator does not necessarily have any of the resource specific roles (depending on whether hierarchic or independent roles are being used). To keep the design consistent, the system resource has a role for every type of resource, controlling resource creation. The system administrator may assign these roles to any user or group.

As the final facet we introduce resource-based groups. Users with a specific role on a resource can be regarded as a group of users. This implicit group of users may be assigned a role for any other resource.

5.3 Future Work

This implementation of resource control leaves the enforcement of access control to the application developers. Unfortunately the enforcement of access control goes across many of the modules of a application, which makes it hard to implement and maintain. *Aspect Oriented Programming (AOP)* could help to handle this in a better way, providing the integration of the access control and management functionality as an aspect in an AOP environment.

While so far we have implemented our model in Java as an RMI-based service, it would be interesting to provide an alternative interface to the service based on the *Java Authentication and Authorization Service (JAAS)* [3]. JAAS is the Java standard API for authenticating users and for assigning privileges, and providing an JAAS-compliant interface to our service would make it possible to integrate it into existing applications which are using JAAS for authentication and access control.

6 Conclusions

The DAC model, which permits granting and revoking of permissions at the discretion of a user, is a very interesting model for the management of permissions in collaborative environments. We think the proposed extensions with groups of users and roles can make this model interesting for resource sharing and collaboration in large organizations.

References

[1] W. K. Edwards. Policies and roles in collaborative applications. In *Proceedings of the ACM 1996 Conference on Computer Supported Cooperative Work*, pages 11–20, Boston, Massachusetts, November 1996. ACM Press.

[2] D. F. Ferraiolo, D. R. Kuhn, and R. Chandramouli. *Role-Based Access Control*. Artech House, Norwood, Massachusetts, April 2003.

[3] C. Lai, L. Gong, L. Koved, A. Nadalin, and R. Schemers. User authentication and authorization in the java platform. In *Proceedings of the 15th Annual Computer Security Applications Conference*, pages 285–290, Scottsdale, Arizona, December 1999. IEEE Computer Society Press.

[4] W. Tolone, G.-J. Ahn, T. Pai, and S.-P. Hong. Access control in collaborative systems. *ACM Computing Surveys*, 37(1):29–41, March 2005.

[5] E. Wilde, S. Anand, and P. Zimmermann. Management and sharing of bibliographies. In A. Rauber, S. Christodoulakis, and A. Min Tjoa, editors, *Proceedings of the 9th European Conference on Digital Libraries*, Lecture Notes in Computer Science, pages 479–480, Vienna, Austria, September 2005. Springer-Verlag.