A Survey of Delegation from an RBAC Perspective

Sylvia L. Osborn, He Wang
Department of Computer Science, The University of Western Ontario, London, ON, N6A 5B7, Canada
Email: sylvia@csd.uwo.ca

Abstract— **Delegation in access control takes place in the presence of an access control model, when a user is unable to perform their administratively assigned permissions. In this paper, we discuss the meanings of the word "delegation" found in the literature, focusing on its use in access control. In the context of a role-based access control model, we survey many possible ways in which delegation can be specified.**

Index Terms— **authorization, delegation, groups, roles, access control**

I. Introduction

Delegation of access rights is a very important function in the business world and in health care. As software systems attempt to support the activities of people doing their everyday tasks, so have systems, particularly access control systems, provided support for delegation. This paper provides an overview of the structural aspects of delegation, using concepts from role-based access control (RBAC) as a framework for the discussion. It complements another survey of delegation by Pham et al. [1], which is less focused on RBAC and which includes implementation concepts.

Access control is that part of a system which controls what users or processes can execute what operations on what objects. In computer security terminology, processes representing users, or sometimes users themselves, are referred to as *subjects*. Access control gives subject s the permission to execute *access mode* m on *object* o. Also important to the overall control of access to objects is *user authentication*, which, as the term implies, is some mechanism by which users log on to a system and are identified. Once users are authenticated and have been given some permissions to access objects, some way of verifying these accesses is required; we will call the process that verifies individual accesses to objects a *reference monitor*. The focus of this paper will be on delegation in access control models, and not on user authentication or reference monitors. Delegation, in turn, takes place when a more permanent access control design proves to be inadequate in some unanticipated situation.

There is a broad spectrum of paradigms in access control. On the one hand we have discretionary access control (DAC), as commonly found in relational database systems and operating systems like Unix, in which objects or data are owned by a user and permission to act on them is given at the discretion of the owner. The other well-known paradigm is mandatory access control (MAC), in which access is based on labels assigned to the subject and object. Basic definitions for DAC and MAC can be found in the glossary of the orange book [2]. Role-based access control (RBAC) provides another way to view access control, and has been shown to be able to simulate both discretionary and mandatory access control models [3]. In role-based access control, users are assigned to roles, which in turn make available a set of permissions.

As well as the different ways of organizing access control just discussed, there is an administrative dimension to access control. Administration can be centralized, as is implied by basic RBAC models [4], [5], or decentralized, as implied by the ownership concept in discretionary access control models. Also, in most systems, one can talk about *design time*, when requirements of different users are accounted for in the design of the system, and the time when the model is deployed and being used, which we will refer to as *run time*. In both of these phases of a system's life, there is a need to make available the permissions required by users to perform their tasks, or to adjust these permissions, in other words to perform *administration*. In the case of discretionary access control, these commands are issued by the owner of the object or by someone who has been given administrative permissions by the owner. In a security model, it may be that administrators cannot themselves perform the actions for which they are assigning permissions to others. For example, in a large company, a complex access control model could be designed so that the administrator assigning new employees to roles is someone from the human resources department, whereas the design of the permission sets of the role is performed by a security expert. Assigning a user to a role is an administrative permission; the administrator holding this administrative permission may not themselves be able to carry out the permissions made available to users assigned to the role.

It may be necessary to alter the design of the access control system after it has been deployed, as the focus of the enterprise changes over time. Thus, administration can take place also at run time.

Delegation, as the term is used in this paper, is motivated by a situation in which one user is temporarily unable to perform one or more of their tasks, because they are too busy, or away from their job due to illness or vacation. The user who normally has certain permissions, called the *delegator*, grants one or more of these permissions (or a role) to a *delegatee*. It is characterized by being temporary, and can be accompanied by a time limit. There are many variations to how many permissions, which permissions are delegatable, etc. which will be surveyed in this paper. Delegation is reversed by a *revocation*.

The following points succinctly summarize the main differences between *administration* of access control and *delegation* of access control:

- The assigning of permissions through administration can

doi:10.4304/jsw.8.2.266-275

take place at design time when the security system is being designed, or at run time, when the security design is being updated. Assigning of permissions through delegation takes place only at run time.
- The person delegating a permission also has the permission. Permissions being assigned by an administrative action can be assigned by someone not having that permission.
- With delegation, the responsibility for the task remains with the delegator. With administration, the responsibility is assumed by the user receiving the permission.
- Delegation is always temporary. Administratively assigned permissions remain permanently assigned until the design is updated.

Many papers have been published describing delegation models. Of course, when a system designer or user of a system needs to specify a delegation, they will use the tools available to them in whatever system they are using. The purpose of this paper is to survey the many ideas that can be found in the literature regarding delegation, and to put them into a common framework so that a student or practitioner can better understand the many variations and details of delegation. The paper proceeds as follows. In Section II we discuss several uses of the term "delegation" in the literature, and arrive at the definition to be used in this paper. Section III summarizes some basic concepts of access control models which are relevant to the sections which follow it. This is followed in Section IV by the main discussion of the paper. First we list characteristics of delegation. This is followed by a detailed discussion of delegation as it can be described in the context of role-based access control. The use of constraints in specifying delegation is then discussed. Other uses of the word delegation in the literature are briefly mentioned in Section V. A summary is given in Section VI.

II. Defining Delegation

The on-line Merriam-Webster dictionary defines delegation as "the act of empowering to act for another" [6]. The term *delegation* has had several uses in computing, all of which relate to this definition. Experienced computer scientists will have encountered the term delegation in object-oriented programming (OOP). Normally, in OOP, inheritance of methods is a compile-time concept. The term delegation is used to describe a run-time phenomenon in which a message to one object can be delegated to another object to execute on its behalf [7]. There are some parallels here with the concept of delegation as used in access control: it is a run-time concept which involves a time-limited passing of control from the delegator to the delegatee, both of which in this case are objects in the execution space of a program.

The idea of delegation, as it is used in access control, follows from practices in the business world. We begin this section by looking at some delegation ideas from business, and proceed to a definition of delegation as it is used in this paper.

A. Delegation in the Business World

In the running of a large organization, managers are encouraged to delegate their tasks to their subordinates. Delegation is distinguished from assignment. When a manager passes along one of their own tasks to a subordinate, this is delegation. The one having the responsibility for the given task is still the manager. When a manager tells a subordinate to do a task as part of the subordinate's normal responsibilities, this is an assignment; the responsibility for the given task is passed to the subordinate [8].

The motivations for delegation in a business include one employee being away either due to illness or vacation, or just being overworked. Delegation can occur on different management levels: the delegatee may be a subordinate, a peer in the management hierarchy of the company, or even someone at a higher level [9].

The following are requirements for delegation in the business world [8]:

CAREFUL DESIGN: The task to be delegated should be carefully explained, including outlining the expected result, how performance will be measured, etc. Both parties have to agree on these.

RESOURCES: The delegating manager should provide the resources needed for the task, such as personnel, equipment and the required access rights to data and information.

ACCOUNTABILITY: The delegatee incurs an obligation after accepting the delegation. He or she is accountable for the proper use of the authority given and the performance of the delegated task. However, from the point of view of management above the delegator, the obligation has not changed. The delegator still is fully obligated to complete the task.

To quote Wikipedia [10], "Delegation, if properly done, is not abdication."

B. Defining Delegation

Delegation in access control occurs in a complex environment where there are potentially many users with complex interactions among them, some of which (e.g. accountability) may be difficult to represent with an automated mechanism. For the remainder of this paper, we will assume the following definition of delegation:

Definition 1: Delegation is the act of one user, the delegator, giving some or all of their assigned permissions to another user, the delegatee, such that the following properties hold:

1) the delegation takes place at run time, not design time,
2) the delegator has the permission(s) at the time of the delegation
3) permissions only are delegated, not user authentication information
4) the delegation is temporary

Note that some aspects of delegation can be part of the security system design, such as specifying policies or constraints, or designing delegatable roles, as we will see below. The giving of permissions through delegation takes place only at run time, whereas administration of more permanent rights can take place at run time if the security design is being updated, and also (normally) takes place at design time.

III. Access Control Models

The basic components of delegation in access control have their origins in the first access control models intended for a business environment. In the earliest models, however, delegation was slightly confused with normal administration of access rights. We thus first consider two traditional access control models, namely the Access Matrix Model and Discretionary Access Control (as it is found in standard relational database packages), before discussing role-based access control, which is used in the rest of the paper.

A. Access Matrix Model

The most basic form of discretionary access control is represented by the access matrix model [11]. Subjects are represented by the rows of a matrix, and objects by the columns. Each matrix entry shows what access rights the subject has on the object, e.g. read, write, execute, etc. One of these rights can be the ownership right, which is also called *control*. The subject with ownership right on object o can alter the entries in the column for object o, i.e. can grant other subjects access to object o. The owner can also assign a copy flag to a specific access right in a matrix entry, which gives the corresponding subject the ability to further grant this access mode on this object to another subject. The basic access matrix model thus has a very decentralized administrative model, since various objects' accesses are managed by different subjects, and also a subset of the rights can be managed by other subjects if these subjects possess the copy flag.

Since the access matrix can be very sparse, it is common to store only the rows, which are then called *capability lists*, or the columns, which are *access control lists* or ACLs.

B. Discretionary Access Control in Relational Databases

Discretionary access control in relational databases is embodied in the GRANT and REVOKE statements, which are part of the SQL standard [12]. The GRANT statement allows a user with control (usually the owner of the table) to grant some or all of the access modes on some or all of the columns of a table to other users. These permissions can be accompanied by the grant option, which allows the grantee to further pass on the permission (similar to the copy flag above). In the original paper describing access control for relational databases, Griffiths and Wade [13] discuss the problem which occurs when a user has been granted a permission in more than one way. Each such grant is at the end of a sequence of grants from the original owner, and these sequences may or may not overlap. When a permission is revoked, each permission whose existence is based on the revoked permission should also be revoked. This concept is called *cascading revoke*. To do this correctly, timestamps must be kept for each of possibly multiple granting actions [14].

In relational databases, there is only one command, the GRANT statement, for assigning permissions to users. If the database administrator initially designs the database, and is the de facto owner of all the tables, then he or she can assign permissions to various users with grant option. These users can then perform a delegation as defined in Definition 1 by using the GRANT statement at run time. On the other hand, a relational database could be designed and created by a collection of users, who retain ownership during the life of the database. If these users grant permission to others, but never do so "with grant option", and never at run time, then the system exhibits administration of permissions but no delegation. Thus it is difficult to distinguish delegation from administration in relational databases because the set of commands for controlling access is so limited. The same confusion exists in all systems exhibiting discretionary access control, where a small number of commands are available to confer access to another user, making it difficult to distinguish between administration and delegation. Role-based access control, as we shall see in the next section, offers a richer model within which to distinguish all the nuances possible in delegation.

C. Role-Based Access Control

Role-based access control models have been discussed since the mid 1990's [4], [15]–[17]. In RBAC, a *role* has a unique name, and has a set of permissions assigned to it. Users are assigned to roles, thus assigning them to a set of permissions in one operation, rather than having to assign the permissions individually. A big advantage to security administrators is that when someone changes their job or leaves the company, they can be removed from roles in one operation, thus having all the corresponding permissions revoked in one operation. RBAC is considered to be "policy neutral", in that it can be configured to mimic DAC, MAC [3] or just to satisfy the requirements of the enterprise, however strict or complex they may be. Ferraiolo et al. [18] contains an extensive discussion of RBAC and its applications. The ANSI standard [19] has grown out of the Sandhu and NIST models.

As well as having permissions and users assigned to them, roles (in hierarchical RBAC [19]) can be arranged in a partial order or role hierarchy, which can be represented as a directed acyclic graph. An example role hierarchy for a wholesale business is given in Figure 1.

The basic components of an RBAC model are users, roles and permissions. In the role graph model [5], each of these three components is potentially represented by a hierarchy or graph. Concerning users, *groups* of users, which are simply sets of users, are also allowed, so that a set of users can be assigned or removed from a role in one operation. Membership in a group can be decided by a security administrator or derived from user attributes, such as being a qualified accountant [20]. There is also a hierarchy of groups determined by set containment, as shown in Figure 2. Individual users are modeled as groups of cardinality 1. In the example group graph, the AccountantsGr group is a subgroup of Everybody, so if Everybody is assigned to role r, all the members of the AccountantsGr set, as well as all the users, are implicitly assigned to r. Groups are not part of the ANSI model. The security designer can use the AccountantsGr to assign them together to a role, or, as we will see later, can use this group to specify potential delegatees in a delegation.

Concerning permissions, implications among permissions can be modeled by several hierarchies or graphs. If the object

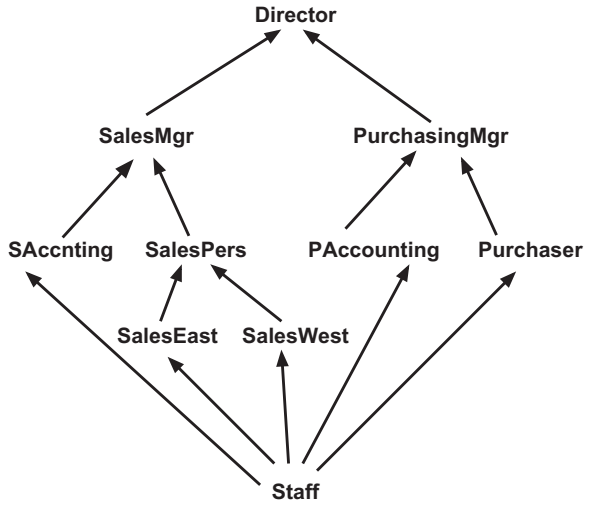

Figure 1. Sample Role Hierarchy

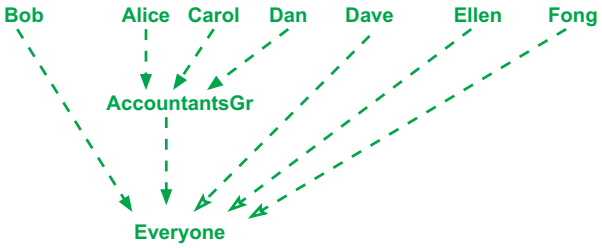

Figure 2. Sample Group Graph

portion of the permission has a complex structure, then having, say, a read permission on a complex object implies having the read permission on all the parts of the object. The object structure which determines how permissions propagate can be shown in an Authorization Object Graph. Figure 3 shows a portfolio of objects representing sales from our warehouse business. The "Sales" object represents a set of objects, each of which is a sale, which in turn contains a number of items. Giving a user read permission on "Sales" would imply read permission on all the contained objects. Implications can also be made concerning the access mode part of a permission. In some systems, having permission to update an object also implies permission to read the same object. These implications can be shown in an Authorization Type Graph. These ideas were originally described for object-oriented databases [21] and have been adapted for RBAC [22].

Concerning roles, we have role hierarchies or role graphs. In such a graph or hierarchy, a directed edge from a junior role to a more senior role represents the fact that the set of permissions available in the junior role is a proper subset of the permissions available in the senior role. Permissions assigned to a junior role are inherited by its senior roles. Users assigned to a senior role are implicitly assigned to its junior roles. In the role graph model, algorithms are given for insertion and deletion of edges, roles, and permissions. One difference between the role graph model and the other models, especially the ANSI model [19], is that in the role graph model, when the

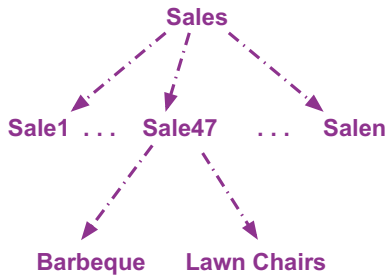

Figure 3. Sample Object Graph

permissions of role r_1 are a subset of those of r_2, a path must exist in the graph from r_1 to r_2 – an edge will be inserted if there is no such path – whereas no such edge will be inserted in the ANSI version.

As well as users, roles and permissions, other components of RBAC models are user-role assignment, also called *membership* in a role, and permission-role assignment. Figure 4 shows permission-role and user-role assignments for our example. Constraints can also be specified to express many additional ideas, such as separation of duty, conditions under which roles may be assigned to a user or under which a user may activate a role, etc. We will not give details for a constraint language, but will discuss later how constraints or rules may be used to govern delegation.

It is customary to consider the role(s) to which a user is assigned through user-role assignment (i.e. the roles the user *can perform*) to be distinct from the role(s) currently activated. In the Sandhu model [4], the concept of a *session* is introduced to model the mapping from each user to the set of roles they currently have activated.

Administration of role-based models was first mentioned in Sandhu et al. [4], where separate administrative roles have permissions in which the object being acted on is a "regular" role in the role hierarchy. The purpose of these administrative roles is to effect the design of the access control for an application. Administrative models for RBAC have been discussed also in [23]–[27].

The components of RBAC, which we will use in the next section to describe different kinds of delegation are, then:

1) roles and the role hierarchy
2) users and user groups
3) permissions and object/access mode hierarchies
4) user-role assignments
5) permission-role assignments
6) sessions
   and
7) administration

IV. Delegation in Access Control

As we saw above for relational databases, it is possible to have delegation in relational databases, or in a similar way to do delegation in any discretionary access control system by making permissions available at run time and then revoking them a short time later. However, all of the dimensions of delegation are best explored in terms of role-based access

Role	Permissions
Staff	read SupplierInfo, write SupplierInfo read CustomerInfo, write CustomerInfo
SalesPers	write Sales
SAccnting	read Sales, write Sales write BankAcct
Purchaser	write Purchases
PAccounting	read Purchases, write Purchases write Cheques

(a) Permission-Role Assignments

User	Role
Alice	SAccnting
Bob	SalesMgr
Carol	PAccounting
Dave	Purchaser
Ellen	SalesPers
Fong	PurchasingMgr
Everyone	Staff

(b) User-Role Assignments

Figure 4. Permission-Role and User-Role Assignments for the example

control, because it is a richer model in terms of describing all the components of an access control configuration.

This section begins by enumerating characteristics of delegation which have been observed by various authors. We then proceed to use the components of RBAC to classify variations on delegation which are possible, with examples. Following this is a discussion of how constraints or rules can enhance delegation.

A. Characteristics of Delegation

In [28], Barka and Sandhu list the following characteristics of delegation:

PERMANENCE: According to Barka and Sandhu [28], *permanent* delegation describes the situation where a user in a role permanently delegates this role to another user. *Temporary* delegation is limited by time. According to Definition 1, what they call permanent delegation we are calling administration of access rights; in this paper we only consider temporary delegation.

MONOTONICITY: *Monotonic* delegation means that the delegator still holds their rights after delegation and can perform this task regularly. *Non-monotonic* delegation means that, after the delegator transfers the rights to the delegatee, they no longer have these rights. However this delegation is not permanent; the delegator can revoke the delegation and get back the rights.

TOTALITY: *Total* delegation occurs when the delegator delegates all permissions of a role to the delegatee, whereas with *partial* delegation, the delegator delegates a proper subset of the permissions of the delegated role to the delegatee.

ADMINISTRATION: The delegation can be performed by the user who is the delegator, in which case this is called *self-acted*, or by an agent, which can be software or another user, which is called *agent acted*. The latter might be necessary if the delegator is unable to access the system on which the delegation is to be performed.

LEVEL OF DELEGATION: *Single step delegation* does not permit the delegatee to further delegate the permission received. *Two- or multi-step* delegation does allow the delegatee to further delegate. This is analogous to the "with grant option" in relational databases, except that here we are dealing only with delegation, and not with administration of rights.

MULTIPLE DELEGATION: Multiple delegation refers to the number of delegatees who can receive the delegation at the same time, i.e. either a single delegatee or multiple ones.

AGREEMENTS: Agreements are the protocols between delegator and delegatee. A *bilateral* agreement is one that both parties of a delegation agree to. *Unilateral* agreement is a one-way decision made by the delegator. In this case, the delegatee has no choice but to accept the delegation.

REVOCATION: Revocation is further classified two different ways: *Cascading revocation* occurs when there is a delegation chain, and the original delegation being revoked causes all delegations along the dependent chain also to be revoked. The other classification is whether or not revocation is *grant dependent* or *grant independent*. The former means that only the original delegator can revoke the delegation, whereas the latter means that any user in a position to be a potential delegator can revoke the delegation.

In addition to these characteristics given by Barka and Sandhu, we can add the following:

STATIC VS. DYNAMIC DELEGATION: *Static delegation* is delegation which can be designed in advance, at design time, whereas *dynamic delegation* is constructed at run time when an unanticipated need arises. The need for this distinction has been independently discussed for Grids in [29], and for role-based access control in [30], [31]. With static delegation, referring back to our example, Alice could decide that, if she is ever away, any member of the AccountantsGr could perform the SAccnting role. If it occurs at run time that no such person is available, and the role is essential, then someone else could be delegated to the SAccnting role with dynamic delegation. Having delegation designed ahead of time, with static delegation, allows consistency checks to be performed in advance [30], [31].

TRANSFER DELEGATION: After a *transfer delegation*, as defined by [32], the delegator can no longer use the delegated permissions. This property is contrary to the assumption we made in Section I, that the responsibility remains with the delegator. Thus, we will not consider transfer delegation

further in this paper.

With the exception of totality, all of these properties deal with how the delegation is managed and carried out. Totality can be described in terms of one of the basic building blocks of RBAC, the role, and whether or not we delegate a whole role or just part of a role.

B. Delegation in Role-Based Models

The basic components of RBAC systems can be used as a framework or reference model to help explore the dimensions of the variations in design of delegation. Recall from Section III-C that these components are: users and user groups, roles and role hierarchies, permissions and object/access mode hierarchies, user-role assignment, permission-role assignment, sessions, and administration. Delegation models using all of these components are surveyed in this section, as well as revocation of delegation. Constraints or rules will are discussed in Section IV-C.

1) Roles and Role Hierarchies: As we saw in Section IV-A, roles can be delegated in their entirety or only part of a role can be delegated. The SAccnting role in our example could be delegated in its entirety, which would be total delegation, or the delegator might wish to delegate only two of the three permissions, say (read Sales) and (write Sales). Total delegation is found, for example, in RBDM0 [33]. If partial delegation is desired, then these "sub-roles" have to be created by the delegator or by an administrator. Let us call such roles *delegation roles*. These delegation roles could form part of the regular role graph or role hierarchy, or could form their own separate hierarchy. In PBDM0 and PBDM1 (two of the delegation models proposed in [34]), the roles to be delegated are regarded as being separate from the role hierarchy, in that the authors do not want any changes to a delegation role to affect any regular roles, and therefore do not want these delegation roles to be junior to any regular roles. In the role graph model, such roles could be placed in the role graph (as long as their permission set is distinct from all the other roles), and the role graph algorithms would add edges to show which roles offer a subset of these permissions and which offer a superset. The restriction being asked for by Zhang et al. [34] is that the designer of the delegation roles not be allowed to add new edges to the role graph, and that a change to a delegation role not affect any regular roles. Partial role delegation is also found in [35] and [36]. An example of partially delegating the SAccnting role in our example, say only including the (read Sales) and (write Sales) permissions, is shown as role Del1 in Figure 5(a), which also shows where the role graph algorithms would place it. In [37], extra permissions are delegated to a role which is already activated; this deals with existing roles, but alters the role at run time to accommodate delegation. Figure 5(b) shows where the SAccnting role with the (write Cheque) permission added, resulting in role Del2, would appear in the role graph.

The role hierarchy is featured in RBDM1 [38] where the can-delegate relation is analyzed in terms of the role hierarchy. A member of role r_1 is not allowed to delegate r_1 or any role junior to r_1 to a user who is a member of a role senior to r_1, since this potential delegatee is already implicitly a member of the junior role. In terms of our example, Ellen would not be allowed to delegate SalesPers (to which Ellen is assigned), or its juniors, to Bob because Bob is assigned to SalesMgr which is senior to SalesPers, and thus Bob can already perform all the permissions in SalesPers and its juniors. RBDM1 does not create any delegation roles. In [36], a richer model of role hierarchy involving inheritance-only and activation-only hierarchies is used to further refine the choice and construction of delegation roles.

The role structure also can be used in expressing revocation. In PBDM0 [34], mention is made of partially revoking a delegated role.

2) Users and User Groups: Users are a primary focus of delegation. It is a user at run time who initiates a delegation, usually to one or more other users. This is the user-to-user delegation spoken of in the PBDMx models [34]. Users can be classified as to whether or not they should ever be a delegatee for a certain role or subrole delegation. This classification can be based on some user attributes, such as qualifications, or perhaps membership in a group or another role. A group can be a dele gatee as in Example 4.1.

Example 4.1: Alice is unable to perform SAccnting. She delegates this role to the AccountantsGr Group, since these users would have the necessary qualifications.

In [39], user attributes are used to decide on role membership and on whether or not a user can be a delegatee; in the latter case this allows the delegator to specify qualifications for delegation, even if they do not know what users might satisfy the qualifications.

The members of a role can be considered to be a group, although only the role graph model includes groups. Membership in one role, say PAccounting, can be used to indicate a set of delegatees, as in Example 4.2.

Example 4.2: Alice delegates role SAccnting to the members of PAccounting.

The delegation in Example 4.2 allows Carol to be delegated. This is a different delegation from Example 4.1, because although Dan is a member of the AccountantsGr group in the group graph, the delegation in 4.2 does not allow Dan to be delegated. This structure is found the role-to-role delegation in [35]. Such a requirement is usually expressed with a constraint, to be discussed below.

In [30], [31], a delegation model is presented based on an administrative model for the role graph model [26]. It makes extensive use of the user/group graph, to model delegators and delegatees. A delegatee group can be set up, and the delegatee can be made a member of this group. Also, delegation chains can be represented by special edges from one delegatee group to another.

3) Permissions and Object/Access Mode Hierarchies: Permissions can be labeled as being delegatable or non-delegatable. Permissions can be added one at a time to delegation roles, as is allowed by PBDM0 and PBDM1 [34], or can be delegated independently of roles [33]. In [37], extra permissions are delegated to an active role, so the unit of delegation in this case is a permission. The object of delegation in [40], which deals with delegation for web-based information

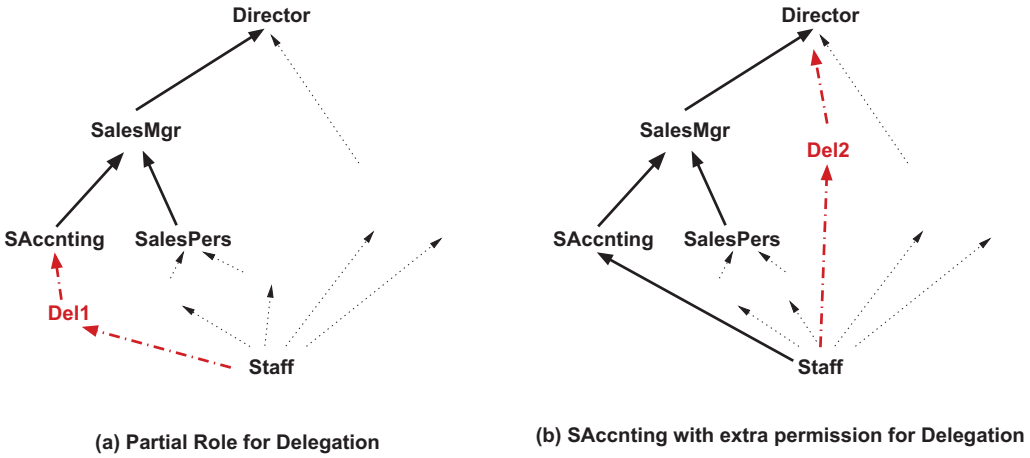

Figure 5. Placement of Delegation Roles in the Role Graph

systems, is any sub-part of a web page to which the delegator has access; in other words the delegation uses the object structure from the object part of the permission in expressing the delegated permission. In our example, Ellen, who is assigned to the SalesPers role, which has one permission, (write Sales), could delegate only one subnode of the Sales object, say Sale42, to someone else, and Sale42 and all its subnodes would be writable by the delegatee.

4) User-Role Assignment: User-role assignment can be used to decide if a user should be a delegatee. A basic assumption made by Barka and Sandhu [33] is that one user u_1 should not be allowed to delegate a role to another user u_2 if u_2 is already a member of the role. This is a property based on the user-role assignment relation. Another way of saying this is to require that the original members and delegated members of a role be disjoint sets. Also observe here that if a user is assigned to a role r_1, they are implicitly assigned to all roles junior to r_1, so there is no point delegating such a user to any of these junior roles; this is another way of stating that a user should not be a delegatee if they are already a member of a role. It is feasible to delegate this user to a role senior to r_1, if they are not a member of this senior role.

In several papers, the term role-to-role delegation is used to describe delegation of one role, say SAccnting, to all members of another role, say PAccounting [33], [35], [41]. This concept is best categorized as a property based on user-role assignment. The delegation has nothing to do directly with role-role relationships, but rather with the membership list of the role PAccounting being used to define possible delegatees. A relationship based on R $\times$ R can be used to specify a can-delegate relation, and the property can also be expressed as a constraint. However, we stress that the fundamental concept being used is the user-role assignment.

Two-step and multi-step delegation, where the original delegatee further delegates to another user, either once (giving two-step) or many times, is also based on user-role assignment; in this case a user who is a member of a delegation role (having been delegated to it) can further delegate the delegation role.

Grant-independent revocation can be defined so that any member of the delegating role can revoke a delegation [33]. This also depends on the original user-role assignment. Of course, grant-independent revocation could also be a permission given to a super-user or administrator, which is based on being assigned to the appropriate administrative roles.

5) Permission-Role Assignment: A delegator could use the permission-role assignment relation to decide which permissions are delegatable, i.e. if the user is a member of a role then the permissions of that role are ones they can delegate. This property is included in PBDM0, and therefore in PBDM1 and PBDM2 [34].

6) Sessions: Recall that a session involves a user and the roles activated for a user at run time. In [37], a medical example is used to motivate delegating extra permissions to a user for a specific instance of an active role. The roles active in a session can be used to decide which roles or permissions are delegatable in either role-based delegation or permission-based delegation. The use of active roles and permissions to specify a delegation at run time is the basis for dynamic delegation as described in [31]. Looking at the example in Figure 5(a), if Alice has activated SAccnting in a session, she could create the delegation role Del1 during this session. Given the user-role assignments in Figure 4, no one is assigned to the Director role, so it is unlikely any of the present users would have a session containing all the permissions of Del2 in Figure 5(b); Del2 could be defined by an administrator using static delegation.

7) Administration: With static delegation, many of the above features of delegation could be designed by an administrator at design time rather than by a user at run time. An administrator could decide on potential delegatees, delegators, design delegation roles, decide on delegatable permissions, etc., all at design time. An administrator can also define the rules or constraints to be discussed below. The system needs to be designed with the required flexibility in place. The most flexibility will occur when delegation can be defined at run time when a truly unanticipated situation occurs. An example of this, in our running example, is if all the accountants are sick, and the delegation has been designed so that only

members of the AccountantsGr group can be assigned to the accountant roles. Without dynamic delegation, the business would not be able to operate properly.

As noted in [30], [31], if the role hierarchy is modified by regular administrative operations on the system, and a role thus affected is a delegation role, then regular administration of the system may affect delegations which are currently in effect. Other examples of administrative operations that might affect delegation are deleting a role, and altering user-role assignments.

8) Revocation: Revocation, as we saw in section IV-A, can be grant dependent or grant independent. When it is grant dependent, the delegator has to issue the revoke. This is also called source dependent revocation in [30], [31]. With grant independent revocation, the revoker can be any user in the delegation role, if it is the same as a regular role, or some administrator or superuser who has this authority. Since we are not studying administrative models in this paper, we will not say much about this except that the permission to revoke a delegation should be part of the permissions given to someone, or revocation will have to rely completely on timeout rules.

When there are delegation chains, authors discuss the ability of some delegatee part way along the chain to be able to revoke their delegation. In this case, the permission to revoke has been given to some or all of the users on the delegation chain. Cascading revocation is a variation of the revocation permission, i.e. the permission is "revoke with cascade" rather than just revoke. The algorithm to handle cascading revocations of permissions, first given in the context of relational databases by Griffiths and Wade [13] and Fagin [14], mentioned in Section III-B, is applicable to cascading revocation of delegation chains. Where there is a delegation chain, and revocation is not cascading, the chain can be shortened by one; for example if the original delegator revokes the first delegation on a chain, this delegator then takes over the position of the first delegatee on the chain [42]. Other forms of revocation mentioned in Bacon et al. [43] are by resignation (the delegatee terminates their session) or by a rule-based system revocation (called constraint violation revocation in [30]), which would occur if some prerequisite required to activate the delegated role is revoked, triggering a revocation of those role assignments based on the prerequisite. Revocation of delegated permissions or roles can also occur at the end of a session [44].

Wang and Osborn [30], [31] introduce a rich model for revocation. If the delegation chain has unlimited depth, one additional form of revocation is to convert it from unlimited depth to limited depth. They also discuss partial revocation, which can be partial revocation of the privileges, partial revocation of the delegatees, or partial revocation of the delegation chain or path, which can take place at the beginning, end or middle of a delegation path.

C. Delegation with Rules

Rules or constraints can be specified for all aspects of RBAC, and can be expressed using many languages and mechanisms. The most common constraints mentioned in terms of RBAC are separation of duty constraints (SOD), which express the fact that one user should not be able to have simultaneous access to certain pairs of roles, objects, etc. because this would allow some fraud to be perpetrated. SOD constraints are also a potential issue for delegation, although other types of rules or constraints are more directly related to the act of delegation.

In this section, we will briefly highlight some aspects of delegation that can be governed by rules, without going into detail concerning syntax for them. Examples of rule specification languages can be found in [43], [45]. Borrowing from Bacon et al. [43], we can classify constraints as being environmental or prerequisite. *Environmental constraints* deal with such things as time, or the location of the users or services. *Prerequisite constraints* deal with things such as membership in another role or group being a prerequisite to being a delegator or delegatee. SOD constraints involving delegation roles are the negative form of prerequisite constraints: a user's membership in another role negates their ability to become a member of a delegation role. Further examples of environmental constraints from [30] are the Absence Constraint, which says that a user can be a delegator only if they are absent from work, and the Workload Constraint (also discussed in [46]) which allows the delegation only when the workload of the delegator exceeds some level.

Orthogonal to this classification, constraints can be based on some value, or can be structural. We use the term *structural constraint* to describe a constraint based on the characteristics of the delegation mechanism itself, such as the depth of a delegation chain, or the cardinality of the user set for a delegation role. A *value-based constraint* is a constraint based on a comparison with some value, which might be an attribute of the user, like the age of the user, or the fact that a certain role is in the delegator's user-role assignment list. An example of a value-based, prerequisite constraint was given in [39] where comparing attributes of users and attributes of permissions is used to decide on delegatees, in a situation where the delegator may not know which delegatees might satisfy the condition. Examples of structural constraints specifying the maximum depth of a delegation chain can be found in [42], [47], [48] and [30]. Other examples of structural, prerequisite constraints from [30] are the Delegatee constraint, which specifies the group to which the delegatee must belong, the Delegated Role Constraint, which specifies a user-role assignment for the delegatee, the Delegation role constraint, which controls which roles can be delegated, and the Maximum Privilege Set Constraint which gives a set of permissions which must not be exceeded for the delegatee for a series of delegations (so that the delegatee does not get too much power).

Rules involving time (which are environmental constraints) can be used in many aspects of RBAC (see for example [49]). As far as delegation is concerned, time constraints can be used to specify the duration of a delegation, i.e. to provide a timeout, or to constrain when delegation can take place, say to limit the time when Alice can be a delegatee to those hours when she is normally at work. Timeouts are mentioned in RBDM0 [33], RBDM1 [38], and [41], [48] and [42]. [50] mentions time predicates that constrain start and end times,

with semantics of "not before" and "not after". In the context of workflow, Atluri and Warner [46] use a time interval to constrain delegation of a task to, for example, the two hours that a delegator is at a meeting.

V. Other uses of the term "Delegation"

In some parts of the literature on access control, it is common to use the term delegation to refer to what we are calling administration in this paper. In discretionary access control, administration of access rights can become very decentralized if the original owner of an object grants rights to other users "with grant option", to use the relational database term. This does not mean that the further granting of rights has anything to do with the motivations for delegation given in Section I, nor with the finer points of our Definition 1. There is no implication that the "delegation" being described is temporary; it is the permanent delegation mentioned by Barka and Sandhu [33] (which we prefer to call administration). Often the word "administration" is used, but instead of saying that the permissions are granted, the authors say the permission is delegated. Some examples of this use of the word delegation can be found in [51], [52]. In a distributed system, furthermore, the whole point is that there is not a centralized authority but that all aspects of the computing environment are decentralized and have decentralized control. Here again is it is common to use the term delegation to refer to granting of permissions, which we would call administration. Examples of this can be found in [53], [54].

On the other hand, in decentralized or distributed systems, a great deal takes place at run time. There is not a concept of design time, but rather users are requesting services from systems they have not previously accessed, in a dynamic environment. In this context, one can talk about the services delegating their administrative functions to another system or agent on the one hand, and, on the other hand, users delegating their capability list in the form of credentials to another service or agent [55]. In the latter case, the delegation that takes place on the user side can deal with identity management as well as capabilities, and is thus often done with the motivation of maintaining the privacy of the user.

VI. Summary

We began this paper by distinguishing between design time and run time, emphasizing that the transfer of access rights through delegation takes place at run time. Several interpretations of the word "delegation" in the computing literature were examined, yielding the definition for the use of the word in this paper. We highlighted the difference between administration of access control and delegation of roles or permissions. Properties of delegation discussed previously in the literature were presented. Using the components of role-based access control, we classified many forms of delegation which have appeared in the literature. This classification is very helpful in exposing the nuances possible in the design of delegation. Various ways in which rules can be used were summarized. Finally, we observed that sometimes, in the literature, the word "delegation" is used to describe what we would call administration of access rights.

VII. Acknowledgements

We thank the following people for thier comments on previous versions of the paper: Neil Croft, Rod Locke, Cecilia Ionita, Xin Jin and Candy Shum. The finanial support of the Natural Sciences and Engineering Research Council of Canada is gratefully acknowledged.

References

[1] Q. Pham, J. Reid, A. McCullagh, and E. Dawson, "On a taxonomy of delegation," *Computers & Security*, vol. 29, no. 5, pp. 565–579, 2010.

[2] DoD, "DOD trusted computer system evaluation criteria," 1985, doD 5200.28-STD, available from http://csrc.nist.gov/publications/index.html.

[3] S. Osborn, R. Sandhu, and Q. Munawer, "Configuring role-based access control to enforce mandatory and discretionary access control policies," *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 2, pp. 1–23, 2000.

[4] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-based access control models," *IEEE Computer*, vol. 29, pp. 38–47, Feb. 1996.

[5] M. Nyanchama and S. L. Osborn, "The role graph model and conflict of interest," *ACM Trans. Inf. Syst. Secur.*, vol. 2, no. 1, pp. 3–33, 1999.

[6] m-w.com, "Merriam-webster online," 2011, www.m-w.com, accessed on August 11.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*. Addison-Wesley, 1995.

[8] M. K. Badawy, "Your toughest tests as a manager - part 4: Delegating without losing control," *Machine Design*, vol. 56, no. 10, pp. 69–73, May 1984.

[9] E. Raudsepp, "Delegate your way to success," *Computer Decisions*, vol. 13, no. 3, pp. 157–8, 163–4, March 1981.

[10] wikipedia.org, "Wikipedia, the free encyclopedia," http://en.wikipedia.org/wiki/Delegation, accessed on Aug 10, 2011.

[11] B. Lampson, "Protection," in *Proc. 5th Princton Symp. on Information Sciences and Systems*. Princeton University, March 1971, pp. 437–443, reprinted in ACM Operating Systems Review, Jan. 1974.

[12] ISO, *Information technology - Database languages - SQL - Part 2: Foundation (SQL/Foundation)*. ISO/IEC 9075-2:2003: ISO, 2003.

[13] P. P. Griffiths and B. W. Wade, "An authorization mechanism for a relational database system," in *ACM Trans. Database systems*. ACM, 1976, pp. 242–55.

[14] R. Fagin, "On an authorization mechanism," in *ACM Trans. Database systems*. ACM, 1978, pp. 310–19.

[15] D. Ferraiolo and R. Kuhn, "Role-based access control," in *Proceedings of the NIST-NSA National Computer Security Conference*, 1992, pp. 554–563.

[16] M.-Y. Hu, S. A. Demurjian, and T. C. Ting, "User-role based security profiles for an object-oriented design model." in *Database Security VI, Status and Prospects*, B. M. Thuraisingham and C. E. Landwehr, Eds. Amsterdam: North-Holland, 1993.

[17] M. Nyanchama and S. L. Osborn, "Access rights administration in role-based security systems," in *Database Security, VIII, Status and Prospects*, J. Biskup, M. Morgenstern, and C. E. Landwehr, Eds. North-Holland, 1994, pp. 37–56.

[18] D. Ferraiolo, R. Kuhn, and R. Chandramouli, *Role-Based Access Control*. Boston: Artech House, 2003.

[19] American National Standards Institute, Inc., *Role-Based Access Control*. ANSI INCITS 359-2004. Approved Feb. 3, 2004.

[20] S. Osborn and Y. Guo, "Modeling users in role-based access control," in *Fifth ACM Workshop on Role-Based Access Control*, Berlin, Germany, July 2000, pp. 31–38.

[21] F. Rabitti, E. Bertino, W. Kim, and D. Woelk, "A model of authorization for next-generation database systems," *ACM Trans Database Syst*, vol. 16, no. 1, pp. 88–131, 1991.
[22] C. M. Ionita and S. L. Osborn, "Privilege administration for the role graph model," in *Research Directions in Data and Applications Security, Proc. IFIP WG11.3 Working Conference on Database Security*. Kluwer, 2003, pp. 15–25.
[23] R. Sandhu, V. Bhamidipati, and Q. Munawer, "The ARBAC97 model for role-based administration of roles," *ACM Trans. Inf. Syst. Secur.*, vol. 2, no. 1, pp. 105–135, Feb. 1999.
[24] S. Oh and R. Sandhu, "A model of role administration using organization structure," in *Proc. 7th ACM SACMAT*, 2002, pp. 155–162.
[25] J. Crampton and G. Loizou, "Administrative scope and role hierarchy operations," in *Proc. 7th ACM SACMAT*, 2002, pp. 145–154.
[26] H. Wang and S. Osborn, "An administrative model for role graphs," in *Data and Applications Security XVII: Status and Prospects, IFIP TC-11 WG 11.3 Seventeenth Annual Working Conference on Data and Application Security*. Kluwer, 2003, pp. 39–44.
[27] D. F. Ferraiolo, R. Chandramouli, G.-J. Ahn, and S. I. Gavrila, "The role control center: features and case studies," in *Proceedings of the 8th ACM SACMAT*. ACM Press, 2003, pp. 12–20.
[28] E. Barka and R. Sandhu, "Framework for role-based delegation models," in *Proceedings 16th Annual Computer Security Applications Conference (ACSAC'00)*, 2000, pp. 168–176.
[29] G. Geethakumari, A. Negi, and V. N. Sastry, "Dynamic delegation approach for access control in grids," *e-science*, vol. 0, pp. 387–394, 2005.
[30] H. Wang and S. L. Osborn, "Delegation in the role graph model," in *SACMAT '06: Proceedings of the eleventh ACM symposium on Access control models and technologies*. New York, NY, USA: ACM Press, 2006, pp. 91–100.
[31] ——, "Static and dynamic delegation in the role graph model," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 10, 2011.
[32] J. Crampton and H. Khambhammettu, "Delegation in role-based access control," *Int. J. Inf. Sec.*, vol. 7, no. 2, pp. 123–136, 2008.
[33] E. Barka and R. Sandhu, "A role-based delegation model and some extensions," in *23rd National Information Systems Security Conference*, October 2000, pp. 396–404.
[34] X. Zhang, S. Oh, and R. Sandhu, "PBDM: a flexible delegation model in RBAC," in *Proc. 8th ACM SACMAT*. ACM Press, 2003, pp. 149–157.
[35] H. Lee, Y. Lee, and B. Noh, "A new role-based delegation model using sub-role hierarchies," in *Computer and Information Sciences - ISCIS 2003, LNCS 2869*. Springer-Verlag, 2003, pp. 811 – 818.
[36] J. Joshi and E. Bertino, "Fine-grained role=based delegation in presence of the hybrid role hierarchy," in *Proc. 11th ACM SACMAT*, 2006, pp. 81–90.
[37] R. K. Thomas, "Team-based access control (TMAC): a primitive for applying role-based access controls in collaborative environments," in *Proceedings of the second ACM workshop on Role-based access control*. ACM Press, 1997, pp. 13–19.
[38] E. Barka and R. Sandhu, "Role-based delegation model/hierarchical roles (RBDM1)," in *Proceedings. 20th Annual Computer Security Applications Conference*, Tucson, AZ, USA, 2004, pp. 396 – 404.
[39] C. Ye, Z. Wu, and Y. Fu, "An attribute-based delegation model and its extension," *Journal of Research and Practice in Information Technology*, vol. 38, no. 1, pp. 3–17, 2006.
[40] S. Chou, E. Lu, and Y.-H. Chen, "X-RDR: a role-based delegation processor for web-based information systems," *Operating Systems Review*, vol. 39, no. 1, pp. 4 – 21, 2005.
[41] L. Zhang, G.-J. Ahn, and B.-T. Chu, "A rule-based framework for role based delegation," in *Proceedings of the sixth ACM SACMAT*. ACM Press, 2001, pp. 153–162.
[42] G.-J. Ahn, L. Zhang, D. Shin, and B. Chu, "Authorization management for role-based collaboration," in *2003 IEEE International Conference on Systems, Man and Cybernetics*, vol. vol.5, Washington, DC, USA, 2003, pp. 4128 – 34. [Online]. Available: http: dx.doi.org/10.1109/ICSMC.2003.1245633
[43] J. Bacon, K. Moody, and W. Yao, "A model of OASIS role-based access control and its support for active security," *ACM Trans. Inf. Syst. Secur.*, vol. 5, no. 4, pp. 492–540, Nov. 2002.
[44] A. K. Mattas, I. Mavridis, and G. Pangalos, "Towards dynamically administered role-based access control," in *DEXA Workshops*, 2003, pp. 494–498.
[45] E. Bertino, E. Ferrari, and V. Atluri, "The specification and enforcement of authorization constraints in workflow management systems," *ACM Trans. Inf. Syst. Secur.*, vol. 2, no. 1, pp. 65–104, 1999.
[46] V. Atluri and J. Warner, "Supporting conditional delegation in secure workflow management systems," in *SACMAT '05: Proceedings of the tenth ACM symposium on Access control models and technologies*. New York, NY, USA: ACM Press, 2005, pp. 49–58.
[47] O. Bandmann, M. Dam, and B. Firozabadi, "Constrained delegation," in *Proceedings IEEE Symposium on Security and Privacy*. IEEE, 12-15 May 2002, pp. 121 – 130.
[48] J. Wainer and A. Kumar, "A fine-grained, controllable, user-to-user delegation method in rbac," in *Proc. 10th ACM SACMAT*. New York, NY, USA: ACM Press, 2005, pp. 59–66.
[49] J. Joshi, E. Bertino, U. Latif, and A. Ghafoor, "A generalized temporal role-based access control model," *IEEE Trans Knowledge and Data Engineering*, vol. 17, no. 1, pp. 4–23, Jan. 2005.
[50] J. Wang, D. Del Vecchio, and M. Humphrey, "Extending the security assertions markup language to support delegation for web services and grid services," in *Proceedings. 2005 IEEE International Conference on Web Services*, 2005, pp. 67–74.
[51] C. Ruan and V. Varadharajan, "A formal graph based framework for supporting authorization delegations and conflict resolutions." *Int. J. Inf. Sec.*, vol. 1, no. 4, pp. 211–222, 2003.
[52] L. Seitz, E. Rissanen, E. Sandholm, B. Firozbzki, and O. Mulmo, "Policy administration control and delegation using XACML and Delegent," in *Grid Computing 2005.*, 2005, pp. 49–54.
[53] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone, "Modeling security requirements through ownership, permission and delegation." in *13th IEEE International Conference on Requirements Engineering (RE 2005)*. IEEE Computer Society, 2005, pp. 167–176.
[54] A. Ravichandran and J. Yoon, "Trust management with delegation in grouped peer-to-peer communities," in *Proc. 11th ACM SACMAT*, 2006, pp. 71–80.
[55] G. Yin, M. Teng, H. Wang, Y. Jia, and D. Shi, "An authorization framework based on constrained delegation," in *Parallel and Distributed Processing and Applications. Second International Symposium, ISPA 2004. Proceedings (Lecture Notes in Computer Science Vol.3358)*, 2004, pp. 845 – 57.