An Administration Concept for the Enterprise Role-Based Access Control Model

Axel Kern
Beta Systems Software AG
Hermann-Heinrich-Gossen-Str. 3
50858 Köln, Germany
axel.kern@betasystems.com

Andreas Schaad & Jonathan Moffett
Department of Computer Science
University of York
York, YO10 5DD, UK
{andreas | jdm}@cs.york.ac.uk

ABSTRACT

Using an underlying role-based model for the administration of roles has proved itself to be a successful approach. This paper sets out to describe the enterprise role-based access control model (ERBAC) in the context of SAM Jupiter, a commercial enterprise security management software[1]. We provide an overview of the role-based conceptual model underlying SAM Jupiter. Having established this basis, we describe how the model is used to facilitate a role-based administration approach. In particular, we discuss our notion of 'scopes', which describe the objects over which an administrator has authority. The second part provides a case study based on our real-world experiences in the implementation of role-based administrative infrastructures. Finally, a critical evaluation and comparison with current approaches to administrative role-based access control is provided.

Categories and Subject Descriptors

K.4.6 [**Operating Systems**]: Security and Protection—*Access controls*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Management, Security

Keywords

Automated identity management, security provisioning, security administration, role-based access control (RBAC), enterprise role-based access control (ERBAC), enterprise roles, administrative role-based access control (ARBAC), scopes, SAM Jupiter

[1]SAM Jupiter was developed by Systor Security Solutions and was recently taken over by Beta Systems Software AG.

SACMAT'03, June 2–3, 2003, Como, Italy.

1. INTRODUCTION

1.1 Background

Role-based access control (RBAC) has established itself as a solid base for today's security administration needs, as documented in the RBAC96 model [14] and the subsequent NIST standard proposal [2]. In particular, *Enterprise Roles* spanning across different IT systems are increasingly used in medium and large organisations as a basis for company-wide security management. An Enterprise RBAC model (ERBAC) has been described in previous papers [5, 4] and was implemented as a core functionality of the commercial security management product SAM Jupiter [10].
The administration of large (E)RBAC systems is a complex task which may be distributed among up to some hundred administrators. To control the authority of administrators, an internal security concept for the administration of the RBAC system itself must be provided. It has been recognised that roles are a suitable mechanism for managing roles, an approach commonly referred to as administrative role-based access control (ARBAC) [12]. Practical examples of how ARBAC may be used to delegate administrative authority have been presented for a database management system [11] and for a legacy access control system [6].
However, to our knowledge there is no other work describing the practical aspects of administrative role-based approaches as we do in the following, particularly with respect to supporting enterprise roles. We believe that this is necessary to:

- assess the validity of current research in this rapidly developing area;
- reconcile the requirements of the actual users of products implementing ARBAC with research;
- provide researchers with feedback in the form of problems encountered by practitioners which should then be addressed more thoroughly.

1.2 Outline

Having given an initial introduction and motivation for our work, the rest of this paper is structured as follows. We briefly describe the enterprise role-based conceptual model of SAM Jupiter in section 2.1. This is followed by a more detailed discussion of how to facilitate a role-based administration approach in section 2.2.1. Our notion of administrative scopes and how these relate to the structure of an

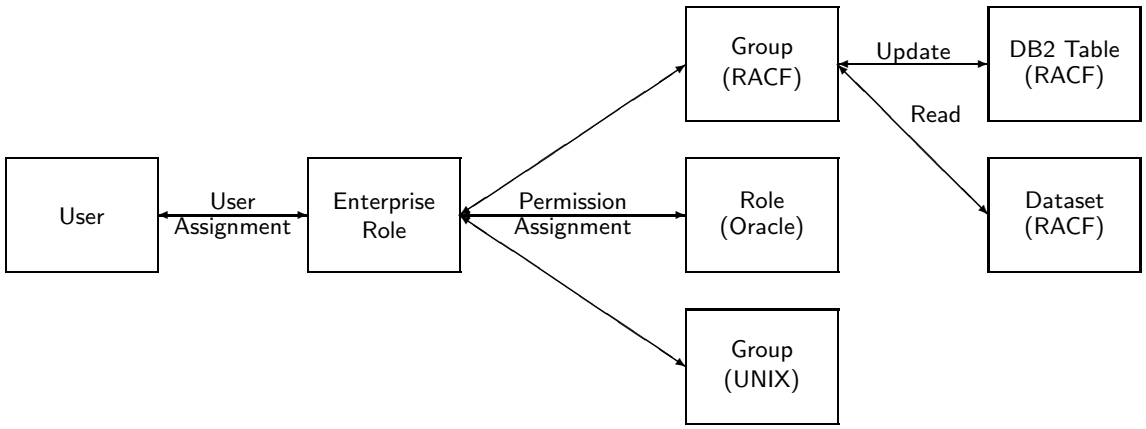

Figure 1: Enterprise Role example

organisation are part of section 2.2.2. We then validate our work by presenting a case study on the implementation of our administration concept in the context of a bank in section 3. The case study is supplemented with further practical experiences and the discussion of some implementation aspects. After having described related work in section 4, we discuss our work in the context of these approaches in section 5.

2. AN ADMINISTRATION CONCEPT FOR ENTERPRISE ROLES

2.1 The Enterprise Role-Based Access Control Model (ERBAC)

In [5] and [4] we introduced the Enterprise-Role Based Access Control Model (ERBAC) which has been implemented in the commercial security provisioning and identity management tool SAM Jupiter [10]. Enterprise Roles allow the administration of users and their access rights across all systems in the IT environment of an organisation. Enterprise Roles span over more than one target system and consist of permissions in multiple systems. These permissions are specific to the target system and can be of various natures. The example in figure 1 shows a role containing a group in UNIX, a role in Oracle and a group in RACF with authorisations for updating a dataset and reading a database table.

Figure 2 shows the resulting Enterprise Role-Based Access Control model (ERBAC)[2]. Enterprise Roles include all permissions needed to perform a specific role. Users are then assigned to these roles. The permissions a user receives through the assignment of a role are propagated to the administered target systems (TS). The Enterprise User definition leads to the creation of user accounts (user IDs) in the TS. A permission can be any operation for an object in one of the underlying target systems. The assignment of a permission to an Enterprise Role does not necessarily cause any update in the target system. The permissions defined for the role are propagated, and the user's accounts receive the associated permissions in the respective TS only when a role is assigned to the user. The process is the same, of course, when permissions are added to or removed from roles.

In addition to the core RBAC features, a general role hierarchy is supported. Enterprise Roles can be assigned to other roles in a directed acyclic graph (DAG). Child roles inherit all permissions from their parent roles (including all permissions that these roles inherit). A user assigned to a child role thus receives all permissions assigned to this role, plus all permissions which the role inherits from its ancestors. Separation of Duty is implemented in ERBAC by rules defining constraints between roles. These rules are evaluated when assigning users to roles and connecting roles to other roles, thus preventing a user from receiving illegal combinations of roles, even in the presence of a role hierarchy.

2.2 Administrative Enterprise Role-Based Access Control (A-ERBAC)

2.2.1 The A-ERBAC Model

ERBAC as described in the previous section is a proven basis for the administration of users and their access rights in medium and large enterprises. The IT infrastructures of such enterprises can consist of some ten thousand to hundred thousand users. To cope with these amounts, companies need a considerable number of administrators[3]. A wide range of business and system knowledge is needed to perform these administrative tasks. These are, therefore, delegated to different groups of administrators.

To allow for this delegation of administrative authority, the ERBAC system itself must implement an administrative security concept. Naturally, this administrative security system is implemented as a target system itself. It uses the same entities as already defined in ERBAC. Administrators are defined as accounts in this target system and receive access rights via roles containing administrative permissions.

[2]For a more comprehensive description of ERBAC and its comparison to the proposed NIST RBAC standard see [4].

[3]Though it is possible to automate a high percentage of the administration tasks (see for example [4]), a considerable amount of manual work still remains.

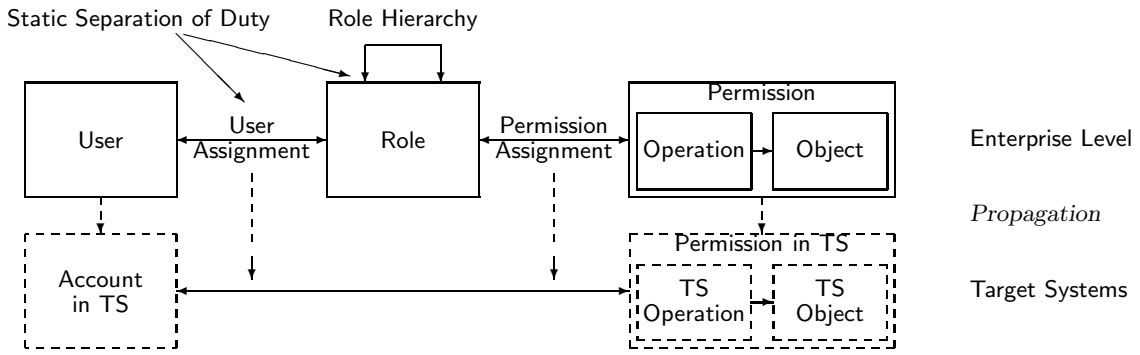

Figure 2: Enterprise RBAC Model (ERBAC)

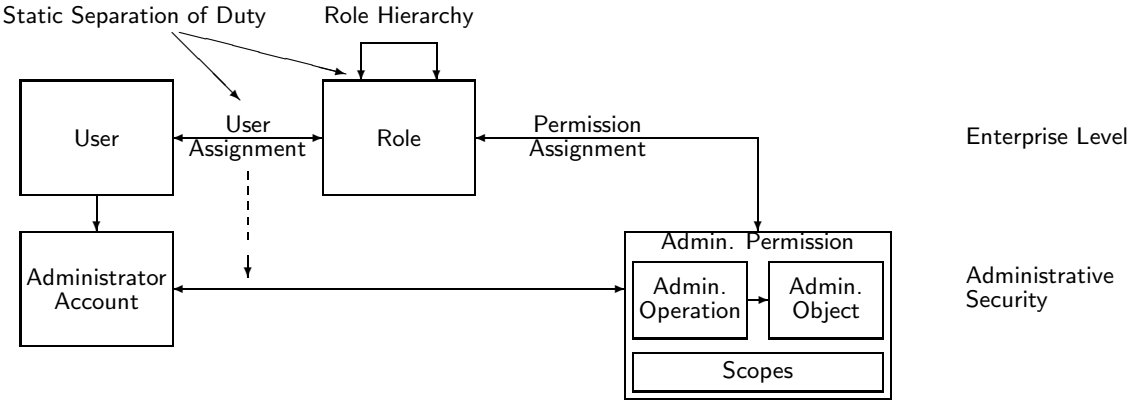

Figure 3: Administrative ERBAC Model (A-ERBAC)

The resulting Administrative ERBAC model (A-ERBAC) is shown in figure 3. In contrast to "normal" target systems, the administrative security system is part of the ERBAC system itself, and assignments need not be propagated to some external target system. If required, Separation of Duty may be enforced by rules as described in section 2.1.

ERBAC Object	Scope
User User-Role Assignment	Enterprise-wide
Role Role-Role Assignment	Enterprise-wide
Role-Permission Assignment Account Permission	target system specific

Table 1: Administrative ERBAC objects

The permissions for the administrative security system consist of operations allowed for the different objects in the ERBAC system. The ERBAC objects are listed in table 1. Users and roles are enterprise-wide entities, whereas accounts and permissions are specific to the target systems. The latter are distinguished for each TS because administrators are often responsible for one or more specific target systems. Relations are considered as separate objects to allow for a more fine-grained authorisation: An administrator who is responsible for building roles may not be allowed to assign users to them.
Administrator accounts and administrative permissions are normal ERBAC objects. Therefore, they can be administered like all other objects by A-ERBAC.

Operation	Abbr.	Scope
`View`	`V`	Object, Attribute
`Insert`	`I`	Object
`Change`	`C`	Object, Attribute
`Delete`	`D`	Object

Table 2: Administrative operations

Table 2 lists the operations which can be specified in administrative permissions. All operations are valid on the object level. In addition, the `View` and `Change` operations can be restricted on the attribute level to prevent administrators from viewing or updating sensitive attributes. This is especially valuable for user and account objects. For example, we can allow administration of a RACF account but forbid change of the SPECIAL attribute providing super administrator rights in RACF. Furthermore, user attributes are

often used to automate assignments of roles to users, so controlling access to them is very important (see e.g. [4] and [1]).
We think that it is important to distinguish between these different operations because in real-life scenarios administrators are normally only allowed to perform specific operations. Some examples include:

- Users are inserted and deleted via an automatic connection to the human resources system. Therefore, human administrators may only be allowed to view and change users and assign roles, but not to insert or delete them.
- A typical local administrator is only allowed to assign roles to users in his department. He may view roles to see which permissions they include, but is not allowed to insert, change or delete them.
- We should not only control update access rights, but also restrict `View` access rights. This is important for two reasons:
  - Security: principle of least privilege,
  - Usability: administration is facilitated if administrators are only able to see the objects they deal with, thus reducing the amount of data they must work with.

2.2.2 Scopes

Objects and operations defined so far in tables 1 and 2 specify only access rights for types of objects. In addition, we must specify the objects themselves, for example which users or roles an administrator may access. When using "classic" schemes, we have the following possibilities:

- We can use an ACL-type scheme and define all users, roles etc. that we want to authorise as administrative objects. Of course, this is not realistic because of the large number of users, roles and permissions.
- We can enhance this scheme by defining profiles as for example in RACF for the objects using naming conventions. This would make administration feasible, but it is a static structure and will cause large administrative efforts if major changes occur in the IT environment.

To mitigate these restrictions, we propose a more flexible approach: the definition of scopes. A scope is an entity which collects objects according to one or more characteristics which may include the organisational structure, a cost center structure or even a combination of several structures. These scopes can then be used to restrict the administrative permissions. Objects are connected to scopes. The scopes themselves are connected in a directed acyclic graph. Figure 4 shows a small example graph built on a cost center hierarchy.
Thus, an administrative permission is defined as a combination of operations, objects and scopes (see figure 3). First, we define which operations may be executed for which objects. We then add a number of scopes for which all these operations are allowed. For each of these scopes the following options may apply:

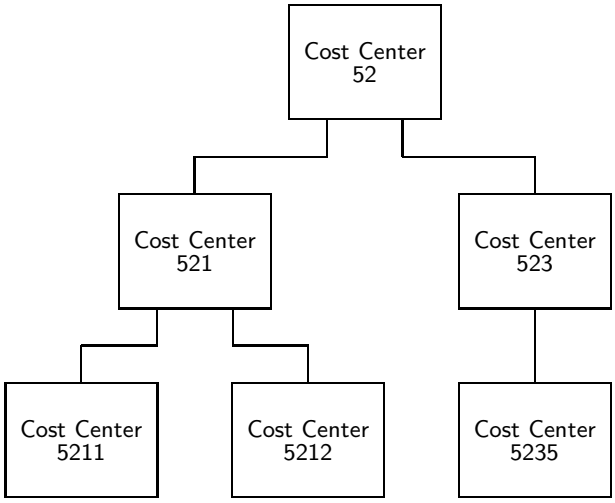

Figure 4: Scope hierarchy

- Node: only the scope node itself is authorised.
- Tree: only the scopes in the sub-tree below the defined node are authorised.
- Exclude: the specified scope (either node or tree or both) is explicitly excluded. This feature allows the simple exclusion of nodes further down in the scope structure. For example, central administrators might be allowed to administer everything except objects in the HR department. They are then authorised for the top scope node, while the HR scope is excluded. Without explicit exclusion, a lot of scopes would have to be assigned.

V	I	C	D	Object
x		x		User
x	x	x	x	User-Role Assignment
x				Role
x				Role-Role Assignment
x				Role-Permission Assignment

Node	Tree	Exclude	Scope
x	x		Cost Center 521
x		x	Cost Center 5212
x			Cost Center 523

Table 3: Example of an administrative permission

Table 3 shows an example permission. It authorises an administrator to view and change users, assign and deassign roles, and view roles and their assignments for the cost centers 521, 5211 and 523 from the scope hierarchy in figure 4.
As administrators can receive several administrative authorisations dealing with the same objects, we have to discuss how to handle contradictions. Different operations on the same object are simply accumulated. Conflicts cannot occur as we do not allow negative authorisations. For scopes

we find a different situation: Scopes build a directed acyclic graph which can lead to multiple inheritance. Taking the three modes "Node", "Tree" and "Exclude" into consideration, we find four cases: direct and indirect specification of a scope node and direct and indirect exclusion which can lead to conflicts. In general, direct grant or exclusion of a scope node always has priority over inherited definitions. In addition, the direct specification of a node prevails over a direct exclusion.
At first glance, scope and role hierarchies seem quite similar. One could also think about using the role hierarchy as a basis for authorising the administrators (as in [12]). There are, however, several reasons which make this approach problematic in real-life scenarios:

- The direction of inheritance may be different for roles and scopes (see also [8]). When building a role tree based on an organisational hierarchy, departments positioned higher up in the hierarchy are normally assigned more general roles. Figure 5 shows an example for an organisational structure. Permissions for all employees (such as use of the mail system) would be assigned to the "Bank" role, region-specific access rights to the "Region" roles and so on. The roles which are lower in the organisational tree inherit the permissions of all roles in the tree above.

  On the other hand, if we take figure 5 as a scope graph, the situation is different. If we implement an administration concept on region level, an administrator assigned to a specific region is allowed to administer it with all included branches but does not necessarily have any rights on bank level. This means that the inheritance direction is reversed.

- The criteria for defining role and scope hierarchies are often different. Scopes are mostly defined based on the organisational structure, whereas roles are often defined by job function. Example: A company defines roles based on job functions such as cashier. An administrator, however, is responsible for a department or division, not for all cashiers in the bank.

- The administration concepts for various ERBAC objects may differ. For example, users might be administered using a decentralised delegation structure based on the organisational hierarchy, roles are administered centrally, and objects and permissions are administered by platform or target system. Therefore, it is important that we are able to implement different administrative permission concepts for these entities. Coupling administrative rights for users and permissions to their assignments to roles is not practical, as has already been shown in [9].

3. PRACTICAL EXPERIENCES WITH ADMINISTRATIVE ERBAC

To validate the model in practice, we shall compare it to the administration concepts of some enterprises. The following examples are based on experiences we have made during the implementation of SAM at large customer sites.

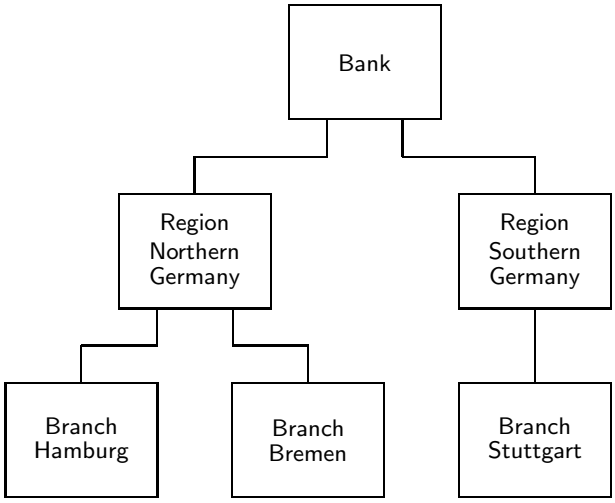

Figure 5: Scope tree based on organisational hierarchy

3.1 A Case Study

A European bank administers about 70 000 users with SAM. Users are created and deleted automatically via a connection to the human resources system (compare also with [15]). The main business roles are also assigned and revoked automatically. The major tasks left to manual administration are the creation and maintenance of roles, the assignment and revocation of additional roles, and password reset.
Administrative roles are created for the main administrator types as described in the following. For each type, we show the administrative permissions in ERBAC assigned to these roles. The delegation concept is based on the various branches of the bank. Therefore, we create a scope hierarchy using these branches. The top node includes all branches of the bank. Figure 5 shows a simplified view of the scope hierarchy.

- *Central administrators* are allowed to administer all ERBAC objects in the system. They are especially responsible for creating and maintaining the roles, but may also carry out all other administrative tasks (see table 4).

V	I	C	D	Object
x	x	x	x	User
x	x	x	x	User-Role Assignment
x	x	x	x	Role
x	x	x	x	Role-Role Assignment
x	x	x	x	Role-Permission Assignment

Node	Tree	Exclude	Scope
x	x		Bank

Table 4: Administrative permission for a central administrator

- *Local administrators* may assign and revoke roles for users, view all access rights of users, and reset their

passwords. These administrative rights are restricted to one branch or a range of branches. As an example, the permission in table 5 contains a branch from the scope tree in figure 5. The `Change` right for users is restricted to certain fields, e.g. the password.

V	I	C	D	Object
x		x		User
x	x	x	x	User-Role Assignment
x				Role
x				Role-Role Assignment
x				Role-Permission Assignment

Node	Tree	Exclude	Scope
x			Branch Hamburg

Table 5: Administrative permission for a local administrator

- *Help desk administrators* may reset passwords for users in one or several branches. The `Change` right is restricted to the password field for users. The example in table 6 shows the permission for a help desk administrator responsible for a region.

V	I	C	D	Object
x		x		User
				User-Role Assignment
				Role
				Role-Role Assignment
				Role-Permission Assignment

Node	Tree	Exclude	Scope
x	x		Region Northern Germany

Table 6: Administrative permission for a help desk administrator

- *Auditors* are allowed to view all ERBAC objects but must not update any object (see table 7).

V	I	C	D	Object
x				User
x				User-Role Assignment
x				Role
x				Role-Role Assignment
x				Role-Permission Assignment

Node	Tree	Exclude	Scope
x	x		Bank

Table 7: Administrative permission for an auditor

3.2 Further Experiences

According to our experiences, it is quite typical that administrative authority is delegated according to some aspect of the organisational structure, as we have described in the previous case study. This is also true for IT centers providing services for several companies and requiring different administrative scopes for every company.

However, the scope structures for user and permission administration often differ. User administration might be delegated according to a departmental structure, whereas permission administration is partitioned according to systems and applications. A good real-world example for this is given in the context of the following organisation, which strictly separates between resource, role and user administration (see figure 6):

1. Resources are logically grouped in so-called business assets. Asset managers build functional roles to which they assign permissions for their resources. They decide which organisational units may use each role.

   A scope is defined for every business asset which contains the corresponding resources and functional roles (see business assets A and B in figure 6). Asset managers are authorised to manage functional roles and assign resources for one or more scopes. In addition, they assign their functional roles to the scopes defining organisational units where they may be assigned. In figure 6, for example, roles A1 and B1 are assigned to department 1, roles A2, B1 and B2 to department 2.

2. Role administrators are responsible for an organisational unit. They build business roles by connecting them to functional roles. The result is a two-level role hierarchy where the business roles inherit the permissions from the connected functional roles.

   A scope is defined for every organisational unit containing the corresponding business roles and users. Role administrators are allowed to manage business roles and assign functional roles for one or more scopes. They are only allowed to view those functional roles which are assigned to the scopes they are authorised for.

3. User administrators are also responsible for an organisational unit. Their job is to assign business roles to users in their unit.

In the previous example, separate scope structures are defined for business assets and organisational units.
Other approaches we have found at customer sites include:

- In many companies the administration of different target systems is performed by different departments. When such a company starts to implement an ERBAC tool, it often does not want to change this organisation, e.g. for political reasons. Thus, administrators remain responsible for their target system(s). As administration concepts differ between the target systems, this might even lead to separate scope trees for them.

- One other bank makes a clear separation between system security and application security. System security consists mainly of the administration of operating system access control (Windows NT, RACF etc.) and is performed by a small number of central administrators without much differentiation of access rights. Application security controls access to the bank applications and allows fine-grained access control. The

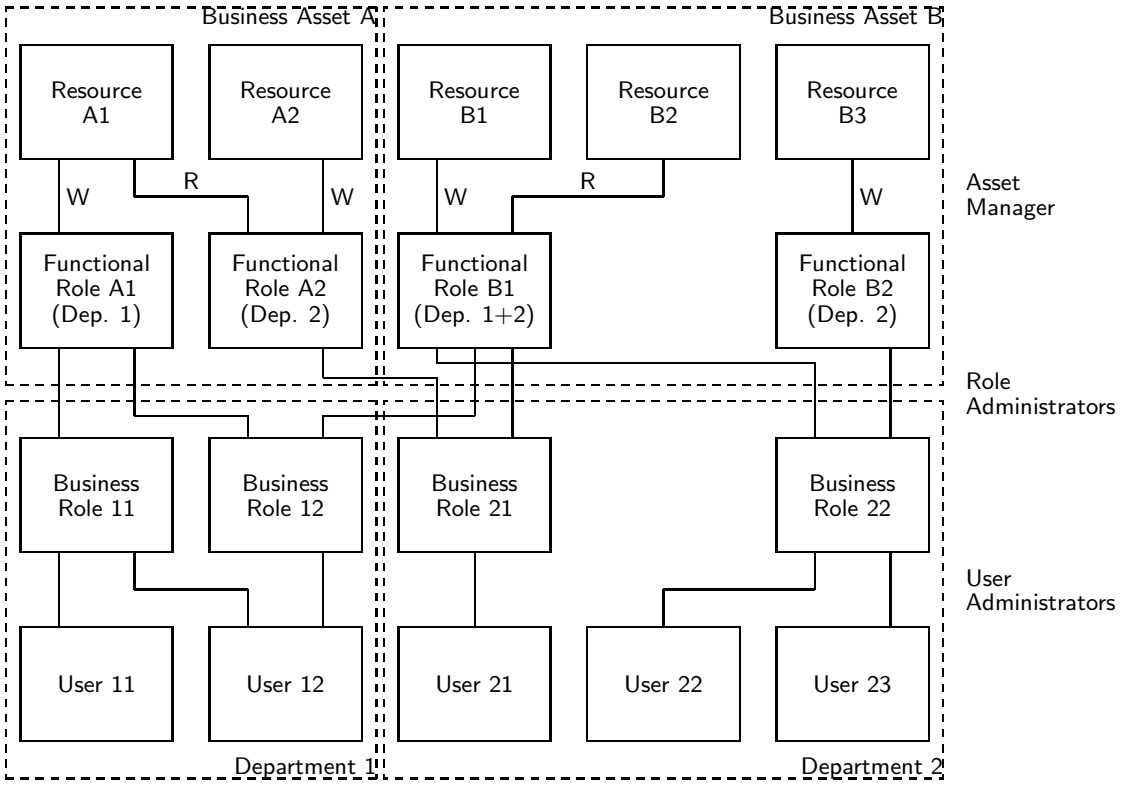

Figure 6: Example with Functional and Business Roles

administration is delegated to the departments, so administrative permissions using an organisational scope hierarchy are defined for the application security target systems.

These examples show that we need a flexible scope structure as defined in section 2.2.2. It is not sufficient to use only the organisational hierarchy alone as a basis for the delegation of administrative authority.

3.3 Implementation Aspects

Besides providing a sound conceptual basis for the administrative security concept, we were forced to consider additional aspects during the implementation of A-ERBAC in SAM Jupiter. These include:

- Every company has administrators who are allowed to do everything in the ERBAC system. For such individuals we have implemented a super administrator flag in the administrative account. Analogously, a super auditor flag allows view rights for every object.
- Several companies have special requirements which do not fit into the general A-ERBAC model. To support such requirements without increasing the complexity of the model, we provide exits in the software which enable the customers to implement these extensions. Examples of such additional requirements might be customer-specific security policies which depend on field contents like allowing only specific administrators to update a field to specific values.
- ERBAC systems must provide good performance. Authorisation checks should not cause long waiting times for the administrators. This is especially critical for `View` authorisations. These must be checked "on the fly" when presenting lists of users, roles etc. in the user interface of the administration tool. However, a system where the data structures are optimised for performance and therefore contain a lot of redundant data, is normally difficult to administer. The solution is to separate the administration and access layer, which is a common practice for application security systems. The administration layer corresponds to A-ERBAC as described above and allows for easy administration. The data of the administrative security system is then denormalised and stored in the access layer in a format which can be checked with minimum delay.

4. RELATED WORK

4.1 The ARBAC97 Model

The administrative role-based access control model (ARBAC97) [12] expresses the idea of using RBAC to manage RBAC through decentralisation of administrative authority, including distinction between regular and administrative roles and permissions. We do not enforce this distinction on a technical level in A-ERBAC, but agree that it is normally made on an organisational level. ARBAC97 consists of three

sub-models. These describe decentralised administration through user-role assignment (URA97), permission-role assignment (PRA97) and role-role assignment (RRA97). Two central concepts of ARBAC97 are the administrative range and prerequisite conditions which regulate and impose restrictions on the administration of system objects. The administrative range reflects the set of roles over which an administrator has authority. Depending on the context, he can assign and remove users to or from a role, alter role hierarchies, and assign or revoke permissions. The authority to control user-role assignments is expressed in a relation $can\_assign \subseteq AR \times CR \times 2^R$. For example, the expression can_assign (ar_x, rr_y,{rr_a, rr_b, rr_c}), would state that a member of the administrative role ar_x can assign a user who currently is a member of regular role rr_y to the regular roles rr_a, rr_b or rr_c. With respect to such user-role assignments, a prerequisite condition could state that any user to be assigned to a role r_1 must already be assigned to another role r_2.

4.2 The ARBAC99 Model

It has been demonstrated that there may be scenarios in which the decentralised administration of a system may be awkward when following the ARBAC97 approach. One example for this is the case of an external consultant assigned to the role "Employee Project X" within a project. Membership of this role might be a precondition for further assignments within the project by the local administrator. The consultant thus automatically qualifies for these possible assignments, and there is no way in URA97 to prohibit further assignments for the consultant.
The ARBAC99 model [13] extends the ARBAC97 model to address such issues, introducing a notion of mobile and immobile memberships of users and permissions in roles. Immobile assignment of a user to a role allows him to make use of the rights associated with that role, however his role membership does not qualify him for any further assignments. Mobile membership on the other hand covers both aspects, access to the permissions of the role as well as the possibility of further role assignments. The problem of the external consultant could thus be easily solved by providing him with an immobile membership to the project role. Mobile and immobile assignments of permissions to roles work analogously.

4.3 The ARBAC02 Model

The ARBAC02 model was introduced in [9], addressing a set of problems that may occur in the administration of user-role and role-permission relationships with respect to the ARBAC97 model. The first underlying reason for these problems is that in ARBAC97 the user and permission pools are dependent on the structure of the role hierarchy. Thus, the concept of an organisational unit, independent of role hierarchies, is introduced as the basis for defining user and permission pools. Assigning a user or permission to a pool is independent from assigning it to a role.
The second identified reason is the top-down approach used in ARBAC97 for permission-role administration. Consequently, a bottom-up approach is suggested in ARBAC02. This means that common permissions are assigned to roles lower in a role hierarchy, while higher roles inherit these and may also be provided with other more specific permissions.

4.4 Policy-Based Systems Management

We introduced the concept of "scope", which appears to subsume the ARBAC02 notion of "pools", in the context of delegation of authority [7]. In this work, scope was defined in terms of domains, which are named sets of principals and resources. The concept of domain provides a flexible and powerful mechanism for capturing many aspects of organisational structure, e.g. cost centers, or departments based on geographical or functional criteria. The security administrator's scope of authority is constrained in two ways:

1. His "Subject Authority" limits the users to whom he can give access rights;

2. His "Target Authority" limits the resources to which he can give access rights.

The motivation for this work was the need to permit security administrators to create access rights, while not possessing such rights themselves. This is achieved by ensuring that the security administrator is not a member of the "Subject Authority" scope.

5. DISCUSSION

As described in the previous sections, there exist several models for administrative role-based access control. These show parallels to the work we have described in this paper. The following discussion shows how ARBAC02 mitigates the main drawback of ARBAC97/99 by using the concept of pools. Our scopes follow this approach, but we argue that A-ERBAC provides a more comprehensive solution.
The evolution from the initial ARBAC97 model over ARBAC99 to ARBAC02 addressed a number of shortcomings. The major problem in ARBAC97 is that the administrative rights for users and permissions are tightly coupled to their membership in a role. ARBAC99 tries to remedy this by introducing the notion of mobile and immobile roles. However, it still uses the role hierarchy as a sole basis for defining administrative authority.
The solution lies in the complete separation of the administrative authority over users and permissions from their membership in a role. This is achieved through the concept of pools in ARBAC02, as well as by using our notion of scopes in section 2.2.2. Scopes are also used in the Tivoli Policy Director, but here in the context of ACLs [3].
ARBAC02 pools and A-ERBAC scopes are principally similar in that they provide different structures for users and permissions which are separated from the role hierarchy. However, our notion of scopes goes further regarding the following issues:

- A scope is deliberately defined as an abstract concept which can, but does not have to be mapped to an organisational structure. While the latter is often used as the basis for the administrative structure, this is not always the case, as our examples show.

- Scopes are also used to define administrative rights for building roles, in contrast to using the role hierarchy. On one hand, this simplifies the administration concept, as all entities are handled in a similar manner. On the other hand, using the role hierarchy may not be

conform with the administration concept of an organisation. This is true, for example, when working with different types of roles as described in our example in section 3.2. Business roles inherit from functional roles, and both types of roles are administered by different groups of people. This separation can be easily expressed by connecting the roles to scopes, but not by using the role hierarchy.

- Not only can we define different scope hierarchies for users and permissions, but we can also do so for all objects in the ERBAC model. This allows for the implementation of fine-grained administrative concepts as is sometimes required in practice. Furthermore, the fact that all objects are treated similarly keeps the administration concept simple.
- By providing several options (node, sub-tree, exclusion) when building scope ranges it is possible to adapt the scope hierarchy easily to a real-world administration structure.
- A persistent issue is how to define who is allowed to administer newly created users. The preferred approach is to import users from a human resources database, thereby putting them into the correct scope. However, as some organisations do not implement such an import interface or not all users are contained in human resources (e.g. external consultants or business partners), users are often inserted manually. This is also well supported by our scope model. If an administrator has the permission `Insert` for user objects for certain scopes, he can create new users in these scopes.

6. CONCLUSION

Enterprise Roles allow the administration of users and their access rights across all systems in the IT environment of an organisation. The Enterprise Role-Based Access Control model (ERBAC) is an extension of RBAC96 based on these Enterprise Roles. The administration of large ERBAC systems is a complex task which may be distributed among many administrators. A common approach to managing RBAC systems is to use roles.

In this paper we have described administrative ERBAC (A-ERBAC), which utilises roles and scopes to allow delegation of administrative authority in ERBAC. A-ERBAC has been implemented in SAM Jupiter, a commercial administration tool. Based on a case study and further experiences during implementation of A-ERBAC in several large organisations, we have shown that it provides a comprehensive and flexible approach for administrative role-based access control.

7. REFERENCES

[1] M. A. Al-Kahtani and R. Sandhu. A Model for Attribute-Based User-Role Assignment. In *Proceedings of the 18th Annual Computer Security Applications Conference, Las Vegas, Nevada, USA*, pages 353–362, December 2002.

[2] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST Standard for Role-Based Access Control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):224–274, August 2001.

[3] G. Karjoth. The Authorization Service of Tivoli Policy Director. In *Proceedings of the 17th Annual Computer Security Applications Conference, New Orleans, Louisiana, USA*, pages 319–328, December 2001.

[4] A. Kern. Advanced Features for Enterprise-Wide Role-Based Access Control. In *Proceedings of the 18th Annual Computer Security Applications Conference, Las Vegas, Nevada, USA*, pages 333–342, December 2002.

[5] A. Kern, M. Kuhlmann, A. Schaad, and J. Moffett. Observations on the Role Life-Cycle in the Context of Enterprise Security Management. In *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT 2002), Monterey, California, USA*, pages 43–51, June 2002.

[6] A. D. Marshall. A Financial Institution's Legacy Mainframe Access Control System in Light of the Proposed NIST RBAC Standard. In *Proceedings of the 18th Annual Computer Security Applications Conference, Las Vegas, Nevada, USA*, pages 382–390, December 2002.

[7] J. Moffett. Specification of Management Policies and Discretionary Access Control. In M. Sloman, editor, *Network and Distributed Systems Management*, pages 455–480. Addison-Wesley, 1994.

[8] J. Moffett. Control Principles and Role Hierarchies. In *Proceedings of the Third ACM Workshop on Role-Based Access Control, Fairfax, Virginia, USA*, pages 63–69, October 1998.

[9] S. Oh and R. Sandhu. A Model for Role Administration Using Organization Structure. In *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT 2002), Monterey, California, USA*, pages 155–168, June 2002.

[10] *For more information about SAM Jupiter see http://www.sam-security.com.*

[11] R. Sandhu and V. Bhamidipati. Role-Based Administration of User-Role Assignment: The URA97 Model and its Oracle Implementation. *Journal of Network and Computer Applications*, 22(3), July 1999.

[12] R. Sandhu, V. Bhamidipati, and Q. Munawer. The ARBAC97 Model for Role-Based Administration of Roles. *ACM Transactions on Information and System Security (TISSEC)*, 2(1):105–135, February 1999.

[13] R. Sandhu and Q. Munawer. The ARBAC99 Model for Administration of Roles. In *Proceedings of the 18th Annual Computer Security Applications Conference, Phoenix, Arizona, USA*, pages 229–238, December 1999.

[14] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, February 1996.

[15] A. Schaad, J. Moffett, and J. Jacob. The Role-Based Access Control System of a European Bank: A Case Study and Discussion. In *Proceedings of the 6th ACM Symposium on Access Control Models and Technologies (SACMAT 2001), Chantilly, Virginia, USA*, pages 3–9, May 2001.