# Advanced Features for Enterprise-Wide Role-Based Access Control

Axel Kern
Systor Security Solutions GmbH
Hermann-Heinrich-Gossen-Str. 3
50858 Köln, Germany
axel.kern@systorsecurity.com

## Abstract

*The administration of users and access rights in large enterprises is a complex and challenging task. Roles are a powerful concept for simplifying access control, but their implementation is normally restricted to single systems and applications. In this article we define Enterprise Roles capable of spanning all IT systems in an organisation. We show how the Enterprise Role-Based Access Control (ERBAC) model exploits the RBAC model outlined in the NIST standard draft[5] and describe its extensions.*

*We have implemented ERBAC as a basic concept of SAM Jupiter, a commercial security administration tool. Based on practical experience with the deployment of Enterprise Roles during SAM implementation projects in large organisations, we have enhanced the ERBAC model by including different ways of parametrising the roles. We show that using parameters can significantly reduce the number of roles needed in an enterprise and simplify the role structure, thereby reducing the administration effort considerably. The enhanced ERBAC features are illustrated by real-life examples.*

## 1 Introduction

The economic and technical changes taking place in today's business world are imposing growing demands for flexibility and efficiency upon enterprises striving to remain competitive. This is true not only for business practices, but also for the IT architectures established to support and implement them. One major issue affecting the competitiveness of an enterprise is the availability and reliability of information. Most companies have realised that protecting corporate data has become a matter of survival. Although information technology itself has been improving significantly over the past decade, the IT security architecture needed to protect such information has not kept pace. This has created a discrepancy in the levels of efficiency and sophistication between the technology and the security mechanisms developed to protect and support it.

Managing an effective IT security environment is becoming increasingly costly and time-consuming for the following reasons:

- Applications and solutions are implemented on an increasing number of different platforms and systems.
- A growing number of different locations are associated with the enterprise, due to company growth or merger activity.
- Discrete and isolated IT security solutions are used across the enterprise. Enterprise-wide security management requires more and more specialised knowledge, making it increasingly difficult to uphold security standards.
- An ever-increasing demand for reliable information security, due to economic as well as technical and legal conditions requiring a tight control on information resources.
- A need for providing users with exactly the access rights needed for daily work when they are needed.

One important strategy pursued by companies in order to remedy this situation is the implementation of an enterprise-wide security administration solution. A number of tools designed to meet this need are available on the market. This article is based on SAM Jupiter, a state-of-the-art security administration tool which

- allows customers to establish corporate standards of security administration valid across all platforms, systems and locations,
- supports central auditing and control,
- provides a uniform administration environment,
- automates security administration,

Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC'02)
1063-9527/02 $17.00 © 2002 IEEE

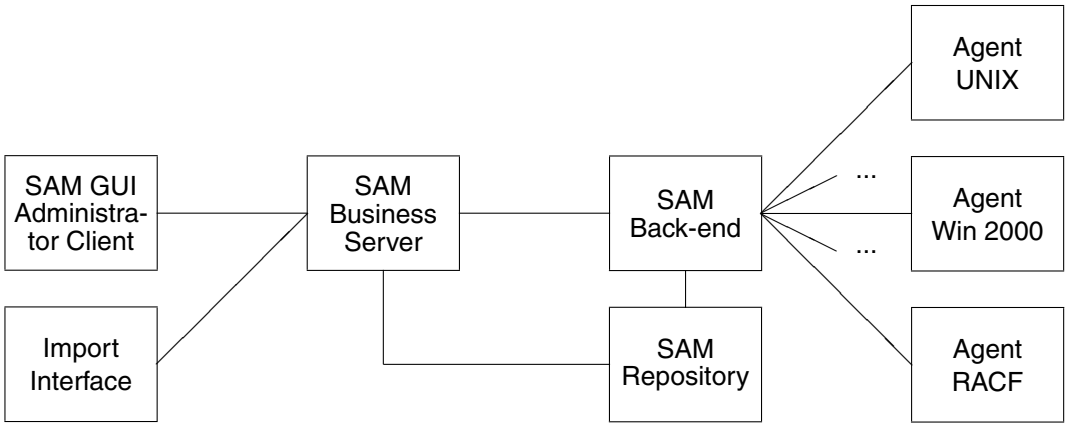


**Figure 1. Architecture of SAM Jupiter**

- supports not only standard security systems, but also has an open architecture able to support home-made systems or systems which will be developed in the future,

- complies with the standards of modern information technology, including the support of central server, client/server and internet/intranet architectures across different communication protocols.

One of the major features of SAM Jupiter is the usage of roles, which we shall discuss in detail in this article. In section 2 we describe the basic functionality and architecture of SAM Jupiter as a basis for this discussion.

Role-based access (RBAC) control has proved to be a solid base for today's security administration needs. RBAC has been a subject of research for many years [3] [4] and is used in a lot of commercial software products. In 2001, NIST proposed a standard [5] that defines the common features of RBAC.

In [6] we introduced Enterprise Role-Based Access Control (ERBAC), the role model used by SAM Jupiter. ERBAC enhances RBAC by the definition of Enterprise Roles, which span different IT systems and form the basis for company-wide security management. After recapitulating the basic components of the NIST RBAC standard, section 3 of this paper describes our ERBAC model in more detail, whereas [6] concentrated mainly on role engineering and the life-cycle of roles. We compare ERBAC with the NIST standard and show how the NIST RBAC levels are mapped to ERBAC. Furthermore, we also present the differences resulting from the fact that Enterprise Roles encompass permissions in a variety of different systems.

In our experiences deploying SAM at a number of large customer sites, we have found a number of drawbacks when using the basic ERBAC model. We have therefore defined and implemented a number of enhancements that have reduced the complexity and administration effort considerably. In section 4, these enhancements are described and illustrated by several real-life examples.

## 2 Enterprise Security Management

The administration of users and their access rights in the IT environments of medium and large companies is a complex and expensive task. Most companies operate a large number of applications running on several different operating systems. According to the Gartner Group, the number and variety of platforms is continuing to grow in most enterprises [14]. As most applications and platforms have their own administration product, this results in high administrative effort and the need for administrators with specific know-how for many platforms.

In this section we describe the architecture and functionality of SAM Jupiter, a commercial enterprise security management product developed by Systor Security Solutions [10]. It is the new generation of the Security Administration Manager (SAM), one of the leading products in this market[1]. SAM Jupiter provides a central point of administration, giving administrators full control of all IT management for employees and resources without compromising on the lowest common denominator of security protection. Interfaces to the specific security systems and applications make it possible to consolidate information in a common security repository using a system-independent conceptual model. When these systems are connected to SAM Jupiter and their data loaded into its repository, administrators work only within the SAM environment and no longer need specific knowledge about the systems to be administered. This not only consolidates the administration work, but also reduces the need for in-depth knowledge about all underlying

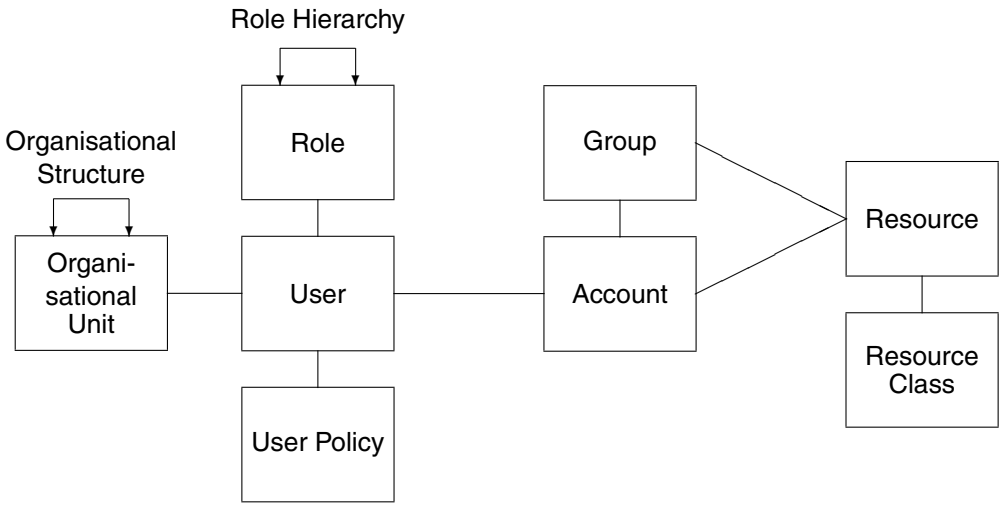


**Figure 2. Entity Relationship Diagram of SAM Jupiter**

systems. All administration work is done in SAM Jupiter and automatically propagated to the underlying systems in the format required.

Figure 1 shows the architecture of SAM Jupiter. It is based on a state-of-the-art 3-tier architecture. The presentation layer is represented by a modern, Web-based graphical user interface which provides access for both central and decentralised administrators. The GUI was developed based on a user-centric development process according to ISO 13407 [2]. By ensuring high usability for the administration interface, it is possible to minimise errors and thereby increase overall security. An import interface is provided for automation purposes.

The SAM Business Server implements the business logic of SAM Jupiter. This is also where the security and administration policies enforced by SAM Jupiter are defined. The SAM back-end component acts as transaction engine for the repository and provides connections to the supported systems – which we refer to as "target systems" – via agents. The agents run on the target platform, propagate the administrative work completed in SAM Jupiter to the relevant security systems and also load the data into the SAM repository. Standard agents are provided for all major software systems. Customer applications can be easily adapted using the SAM connector technology. Specific connectors are provided for supporting application security systems, LDAP-based systems, and other company-specific applications.

Figure 2 shows the basic entity relationship diagram of SAM Jupiter. The left-hand side of the diagram shows the enterprise-wide entities, positioning the user as the central entity. A user can be a member of an organisational unit. Organisational units are connected to build an organisational structure. Permissions can be assigned either via roles – which is the recommended method – or explicitly. On the right-hand side, the target system specific security entities are mapped: Users receive accounts (also called user IDs) in a target system. Accounts can become members of groups. Both accounts and groups can be authorised to resources, which are often categorised into different classes (such as database, table or view for a database system). All this data is stored in the SAM repository as the basis for administration and review functions. Administration performed in SAM Jupiter is directly propagated to the target systems. The review functionality is symmetric and allows viewing permissions from the user side as well as from the resource side – independent of the way the target system actually stores the information.

To further reduce administration costs, most enterprises wish to automate administration. The most accurate information about its employees can often be found in the human resources database. Extracted information such as employee number, organisational unit, location or job description can be used to add and delete users automatically as well as update their access rights. A prerequisite for automation is the usage of roles corresponding to organisational structures, job descriptions etc. (see section 3.2). The mapping of user attributes to the roles a user will receive is achieved by defining a set of rules.

If a new employee starts with the company, this information is transferred directly from the human resources database to SAM Jupiter, which automatically transforms the information to role assignments and makes the corresponding updates in the connected target systems. In addition, when an employee leaves the company, all of the employee's accounts and access rights are automatically

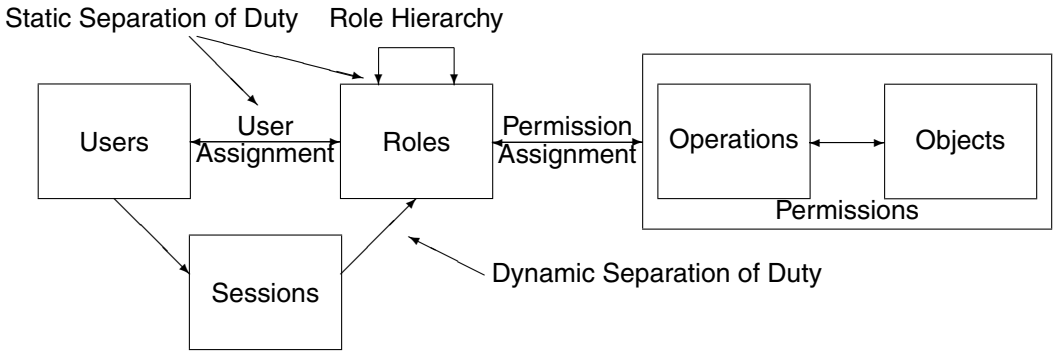


**Figure 3. Standard RBAC Model [5]**

deleted, thus greatly reducing security risks.

Consistency between the information in the human resources system and SAM Jupiter can be ensured by extracting all relevant users from the human resources database, evaluating the corresponding rules and comparing the results with the current status in the SAM repository. The differences are then adjusted accordingly. The same procedure is used when rules change, for example, due to organisational changes.

## 3 Enterprise Roles

### 3.1 Role-Based Access Control (RBAC)

Roles are a powerful concept for simplifying access control. In Role-Based Access Control (RBAC), permissions are not directly associated to users but are instead collected in roles. Users are then assigned to these roles, thereby acquiring the roles' permissions. A role normally contains all rights needed in an organisational unit or for a specific job function [12].

The usage of RBAC offers numerous advantages for an enterprise:

- Separation of responsibilities: The business processes "Defining a role" and "Assigning a role" are separated. The definition of roles, which requires system-specific know-how about resources, is performed by system or security administrators. On the other hand, the assignment of roles to users is done in the business departments by people who know which job function a user performs, but do not have system know-how.

- The number of administration tasks can be reduced, as the number of roles is normally considerably lower than the number of users and permissions. (Example: A large German bank has defined 400 roles for 40,000 users.) This reduces the administration costs significantly.

- RBAC allows a better overview of the permissions granted to a user. On one hand, auditors can more easily see what access rights users have and check whether they are supposed to have them. On the other hand, administrators can authorise users in a more controlled way. In companies working with individual permission assignments, users often accumulate access rights when changing positions within the company. Nobody really knows anymore which of these rights belong to older job functions and which of them the employee really needs. The usage of roles connected to such functions thus increases security.

- Last but not least, roles are an important prerequisite for automating security administration.

In recent years, roles have been implemented in many commercial systems. However, there is no widely accepted RBAC model, meaning that implementations differ considerably. In 2001, an RBAC standard was proposed which defines the core and extended capabilities of roles [5].

*Core RBAC* defines the basic functionality of roles. It includes sets of five basic data elements: users, roles, sessions, objects and operations (see figure 3). Roles collect permissions for objects (permission assignment). Users can be assigned to roles (user assignment). Both types of assignments are many-to-many relations. During a session, a user can activate one or more of the assigned roles. Each session is associated with one user, whereas a user can have several sessions at the same time.

*Hierarchical RBAC* extends core RBAC with role hierarchies that allow the structuring of roles to correspond with functional or organisational hierarchies. Child roles inherit all permissions of their parent roles. The standard differentiates between general and limited role hierarchies. *General*

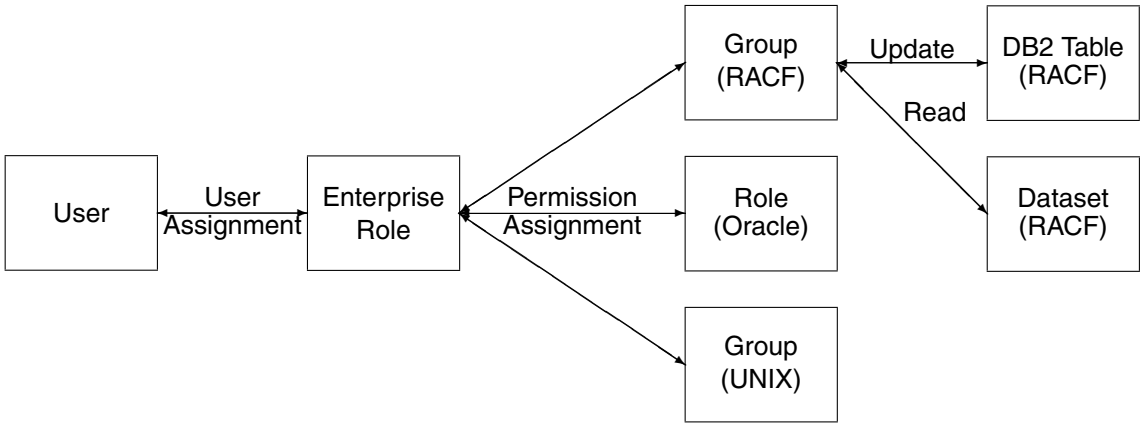


**Figure 4. Enterprise Role Example**

*role hierarchies* allow roles to be connected in an arbitrary partial order, whereas *limited role hierarchies* are restricted to tree structures.

*Constrained RBAC* adds Separation of Duty (SoD) relations to core RBAC. The standard allows for both static and dynamic SoD. *Static Separation of Duty* enforces constraints on assignments of users to roles. For example, two roles can be defined as mutually exclusive. In contrast, *Dynamic Separation of Duty* limits the activation of roles for a user's session.

In addition, the standard contains requirement specifications for administrative functions, supporting system functions and review functions. *Administrative functions* enable administrators to create and delete the RBAC options and their relations. *Supporting system functions* are used to maintain sessions. *Review functions* provide reports such as "All permissions of a user".

### 3.2 Enterprise Role-Based Access Control (ERBAC)

As already mentioned, the IT environments of large enterprises consist of a variety of platforms and applications. These include a number of operating systems (e.g. OS/390, Windows NT/2000 and UNIX), databases (e.g. Oracle and DB2), standard applications, such as SAP's R/3, and a large number of business applications. Most of them have built-in security components (as in the case of Windows NT or Oracle) or are secured by a separate security product (such as RACF, ACF2 or Top Secret for OS/390). The mechanisms used by these security components differ significantly. Some of them already work with roles.

A typical enterprise user must have access to a variety of systems and applications on different platforms. However, existing role concepts are mainly specific to particular applications and operating systems, causing administrative overhead. For example, to access an application, a user may need parallel sessions at different layers, such as Windows NT, OS/390 and a database, requiring access rights for all of them. These rights must be administered separately in the participating systems as no common RBAC support is currently provided. There are some developments towards broadening the scope of operating system security. Operating systems such as Netware and Windows 2000 allow administration components of other systems and applications to use their directories. However, these approaches are restricted to only a subset of the systems operated by a typical enterprise; in particular, the support of mainframe and midrange systems is very restricted.

Because of the variety of different systems in the market and their dynamic behaviour, modern enterprises require a more comprehensive RBAC solution. Such a solution is provided by introducing Enterprise Roles, as implemented in SAM Jupiter. 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 4 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 5 shows the resulting model, which we call the Enterprise RBAC model (ERBAC). It is based on RBAC96 [12] and the NIST role standard draft [5] (see figure 3). The basic ERBAC model is analogous to core RBAC. Enterprise Roles collect all permissions needed to perform a specific role. Users are then assigned to these roles. The main difference between ERBAC and the RBAC96 model lies in the notion of sessions. In an enterprise-wide administration concept, all systems in the enterprise are administered, but without control of the actual user sessions. Therefore, sessions cannot be part of ERBAC. Instead, the permissions a user receives through the assignment of a role are prop-

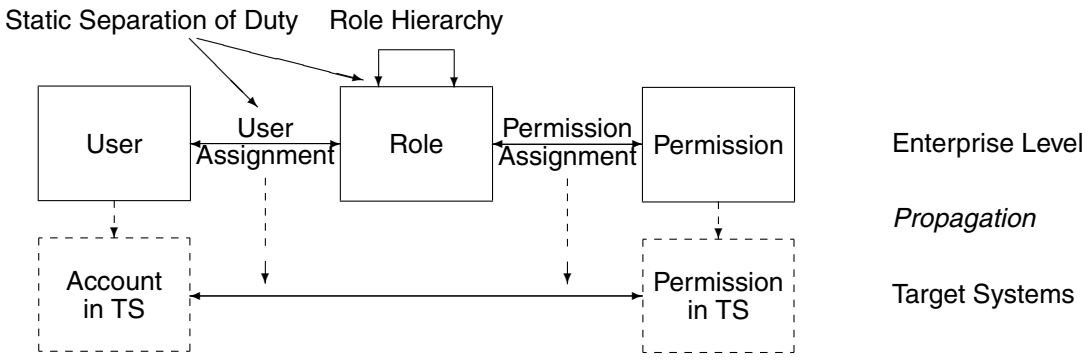


**Figure 5. Enterprise RBAC Model (ERBAC)**

agated to the administered target systems (TS). The Enterprise User definition leads to the creation of user accounts (user IDs) in the target system. A permission can be any authorisation (called an operation in core RBAC) to a resource in one of the underlying target systems. The assignment of a permission to an Enterprise Role does not necessarily lead to any update in the target system. The permissions of 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 same happens, of course, when permissions are added to or removed from roles.

As already mentioned, a permission can be any entity in the target system, such as a group, a role or an authorisation. Let us take a deeper look at the concept of groups: There has been some discussion about the differences between the notion of roles and groups (e.g. in [11]). Groups can be seen more as grouping mechanisms for users (see e.g. [8]) or as groups of permissions. In our context, a group is simply a target system entity which bundles permissions in the target system. From the ERBAC viewpoint, a group is therefore a type of permission that can be assigned to Enterprise 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. 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. Role hierarchies allow easy structuring of roles and reduce redundancy. This leads to a smaller number of roles to be defined in an enterprise and less administrative effort. However, experience has shown that the role tree should not be too deep, as it can otherwise become very difficult to maintain. A role hierarchy with a maximum of three to four levels is recommended.

The RBAC standard draft also defines constraints. *Static Separation of Duty* is implemented in ERBAC by rules defining constraints between roles. These rules are evaluated when assigning users to roles and roles to roles, thus preventing a user from receiving illegal combinations of roles, even in the presence of a role hierarchy.

As an ERBAC system does not control the actual sessions of a user, it cannot directly enforce *Dynamic Separation of Duty*. Instead, it must rely on such mechanisms in the target systems. A common way to enforce dynamic SoD is to use different accounts for different tasks. In ERBAC, we can simply define different users for these tasks. Thus, by defining static Separation of Duty constraints in ERBAC, dynamic Separation of Duty can be enforced in the underlying systems. If the target system itself supports dynamic SoD, this feature can be controlled directly by ERBAC.

In addition to the role model, the RBAC standard also defines administrative and review functions. Our implementation of ERBAC in SAM Jupiter also offers a complete set of administrative commands for all supported functions. A complete list of all supported commands, however, goes beyond the scope of this paper. The review functions are based on the ERBAC repository and are realised using a Web-based reporting engine. The symmetric nature of the repository allows for a comprehensive set of lists including basic and advanced lists for all ERBAC levels. Some examples for advanced reports are:

- List all permissions of a user (including those inherited from all directly or indirectly connected roles).
- List all users to which a role is connected directly or indirectly via the role hierarchy.
- List all users with a specific permission.

## 4 Enhanced ERBAC

ERBAC as defined in the previous section provides a good basis for user and security administration. However, our experience during deployment of roles with SAM at several customer sites showed that the sole usage of this

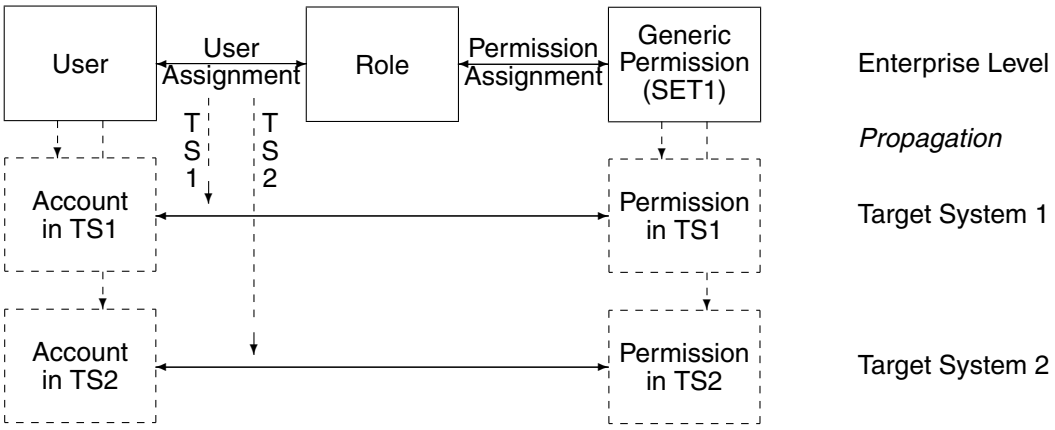


**Figure 6. ERBAC with Generic Roles**

model would have led to a large number of roles and thus to a high administration effort. There are basically two reasons for this: multiple factors defining roles and the need for fine-grained control of application security.

The access rights a person receives are normally based on a number of factors. These may be organisational unit, job, location or others. As the combination of these factors defines the rights, one cannot simply build separate role hierarchies based on organisation, job etc. Instead, a role must be defined for every valid combination of these factors. The resulting role structure would obviously be very complex and difficult to maintain[1].

In typical business applications, fine-grained, restrictions to access rights often apply. For example, different loan managers may be allowed to approve loans up to different amounts. Using the described ERBAC model, one loan manager role must be defined for every different maximum approval amount. This would again lead to many similar roles differing only in a single constraint.

The solution for these problems is to parametrise roles. We have therefore enhanced our ERBAC model with attributes and rules. Attributes can be assigned to the following entities:

- users,
- roles,
- user assignments,
- permission assignments,
- role-to-role assignments.

These attributes may specify constraints or other values relevant for access control decisions. Rules specify what happens when attributes are changed or assignments are given or removed.

In the following sections, we describe how enhanced ERBAC can dramatically reduce the number of roles, thus greatly facilitating role administration. All features are motivated by real-life situations that we have encountered during role deployment in large organisations.

### 4.1 User Attributes

The user in our ERBAC model contains a rich set of standard and company-specific attributes. These attributes are used for a number of important functions:

- A set of attributes such as name, title, telephone number describes the user. Furthermore, they can also be propagated to a target system when an account is created for the user.
- Several attributes describe the user's organisational unit, job function(s) and so on. These attributes provide the basis for automation of user administration as they normally define the roles a user will receive. As described in section 2, this data is often extracted from a human resources system. If these attributes change, the roles a user receives or loses are computed using rules and automatically assigned or deassigned. Of course, automation can considerably reduce administration costs and is therefore the main goal of many companies when implementing an enterprise-wide user administration tool. Some companies have

[1] The situation is comparable to multiple inheritance in object-oriented programming. Extensive use of – especially multiple – inheritance leads to unmaintainable software systems.

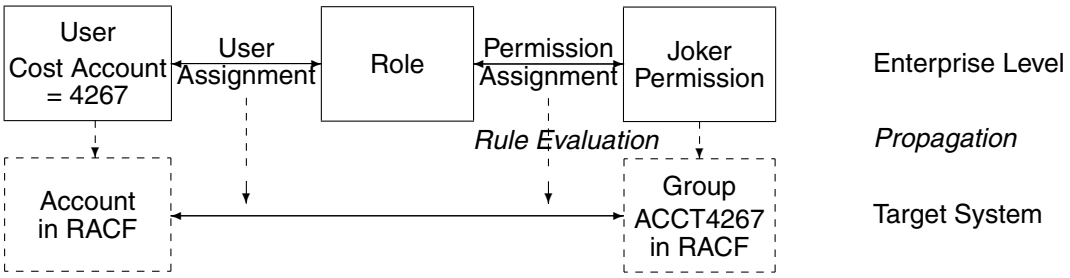


**Figure 7. ERBAC with Joker: Example**

succeeded in automating more than 90-95% of their user administration tasks. As an example, table 1 shows some figures for role-based administration in a European bank.

| |
|---|
| 40 000 users |
| 12 000 changes of user assignments to roles per week (fully automated) |
| 600 changes of permission assignments to roles per week (manually) |
| → 95% automation of administration |

**Table 1. Figures for Role-Based Administration in a Bank**

- A further possibility is to use user attributes for specifying user-specific information which can be used as constraints for roles and permissions or for other administration tasks. The following sections go into more detail on this.

### 4.2 Generic Roles

For several types of systems – such as Windows NT and UNIX – many organisations have a number of locally distributed installations. A user is defined in one or more of these systems according to location (or some other attribute). Users working at more than one location may be defined in several systems. Typically, the group and permission structures of these systems are defined quite similarly.

To prevent building separate role structures for all of these systems, we added the concept of roles with generic permissions to ERBAC. Normal roles are collections of permissions defined in specific target systems. Generic roles allow the assignment of generic permissions defined for a set of target systems. When such a role is assigned to a user, one or more target systems from this set are specified. The user then receives these permissions only in the specified target systems.

Figure 6 illustrates this feature. A generic permission defined in a target system set SET1 is assigned to a role. The target system set SET1 may contain five target systems with similar permission structures (TS1 ... TS5). When assigning our role to a user, we specify the target systems TS1 and TS2 from this set. The user then receives the permissions defined in the role for these two systems.

### 4.3 Joker Permissions

Information from several structures is often needed to define a role. A common example is that a user's access rights depend on location and job function[6]. If we tried to build a role tree including both factors, the tree structure would become quite complex. It also would contain a large number of roles, as one role is needed for every valid combination of location and job function.

We have established a successful alternative approach that avoids increasing complexity. Only one structure is used as a basis for the role graph, while the other is implemented via parameters of these roles. As an example, we can take the job function for building the role graph and define the location as an attribute of the user/role relationship.

This feature is implemented by assigning so-called joker permissions instead of explicit permissions to a role. When assigned to a user, the actual permission is computed using the attributes of the user and/or user-role connection on the basis of rules. The permission is then granted to the user. The rules use attributes of the user (e.g. organisational unit, location, job function) to compute the actual permission. A naming convention for the permissions is a prerequisite for this method.

Figure 7 shows an example for the usage of jokers. In a bank, all users receive a membership to a RACF group on the mainframe according to their cost accounts, which are

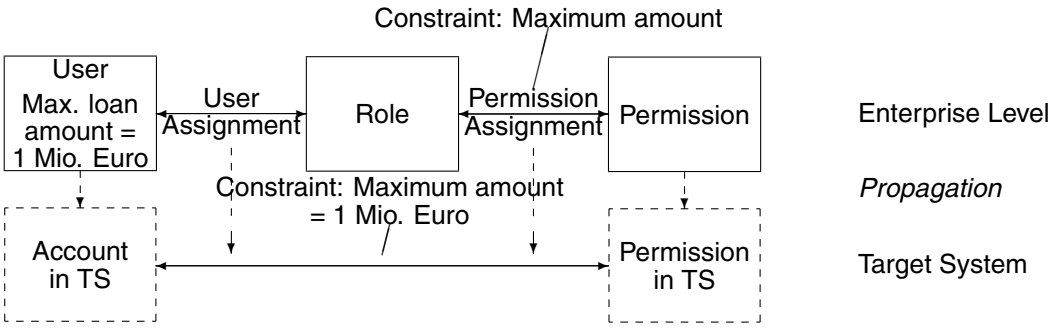


**Figure 8. Example for user-specific constraint**

represented by four digit numbers. For this purpose, groups in RACF are defined with names consisting of the string "ACCT" followed by the account number. A role is now defined containing a Joker Group. When a user is assigned to this role, a rule is triggered that computes the name of the group by concatenating the fixed string "ACCT" with the cost account of the user (taken from the user attributes) and assigns the user in RACF to this group. This mechanism is quite powerful, as it uses information about the user and the permission structure to automate the administration process.

### 4.4 User-Specific Constraints

User-specific constraints constitute a further aspect often occurring – especially in business applications. People doing principally the same job may have different restrictions. Some examples include:

- A bank teller may only work with a specific set of customer accounts.
- A loan manager may grant loans up to a specific amount.

It would be possible to build separate loan manager roles for every different maximum amount or separate bank teller roles for every range of customer accounts. Obviously, this is not a good solution as it would lead to a large number of similar roles.

In ERBAC we have enhanced the permission assignment with additional parameters. The underlying application security system already has parameters for its authorisations in order to check the constraints. These system-specific parameters are now mapped to permission assignments. Furthermore, rules are defined to determine how these parameters are filled. This can be anything from filling the parameter with an attribute from the user record to complex computations. When a user is assigned to a role in ERBAC, these parameters are computed and propagated to the underlying system.

Figure 8 shows a simple example of user-specific constraints. A banking application has defined the constraint "Maximum amount" for the assignment of a permission to approve a loan. A loan manager role is now built that includes this constrained permission but does not define an explicit amount for the constraint. A user has an attribute "Maximum loan amount" of one million Euro, which defines the limit. When this user is connected to the role and the permission is propagated to the target system, the constraint is filled with the value from the user attribute.

## 5 Conclusion

Role-Based Access Control is an effective mechanism for simplifying the administration of users and access rights in complex IT infrastructures. To support enterprise-wide security management, we have introduced Enterprise Roles and the Enterprise Role-Based Access Control Model (ERBAC). ERBAC has been implemented in the commercial security administration tool SAM and has proven successful in many projects in large organisations. We have also shown that enhancing ERBAC with parameters reduces the number of roles dramatically, thereby minimising administration and role maintenance costs.

Future work will be done to improve the ERBAC model and its validity will be proven in deployment at further customer sites. In particular, we must cope with the challenges deriving from the growing distribution of IT systems over the internet.

A second important area is role engineering, as the deployment and maintenance of roles requires a thorough process. We have already defined a role life-cycle in [6]. The role-finding process in particular will be further investigated. So far, two approaches have been considered. On one hand, a top-down approach starts with the business structures and processes and refines them to obtain roles (see

also [9]). The bottom-up approach, on the other hand, takes the existing permissions and applies data mining techniques to find clusters of permissions which represent roles. It will be interesting to combine both approaches.

## References


[1] R. Awischus. Role-Based Access Control with the Security Administration Manager (SAM). In *Proceedings of the Second ACM Workshop on Role-Based Access Control, Fairfax, Virginia, USA*, pages 61–68, November 1997.

[2] A. Beu, A. Kern, and J. Schwagereit. "Das User Interface ist wunderschön...". Der benutzerzentrierte Gestaltungsprozess nach ISO 13407 in der Praxis. *Java Magazin*, pages 28–35, May 2002.

[3] B. Biddle and E. Thomas, editors. *Role Theory: Concepts and Research*. Robert E. Krieger Publishing, New York, 1979.

[4] D. F. Ferraiolo and D. R. Kuhn. Role-Based Access Control. In *15th NCSC National Computer Security Conference, Baltimore*, 1992.

[5] 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.

[6] 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.

[7] A. Kern, M. Kuhlmann, and R. Wick. Ein Vorgehensmodell für Enterprise Security Management. In *Proceedings of the Working Conference on IT Security in Electronic Business Processes, St. Leon-Rot, Germany*, September 2002.

[8] S. Osborn and Y. Guo. Modelling Users in Role-Based Access Control. In *Proceedings of the Fifth ACM Workshop on Role-Based Access Control, Berlin, Germany*, pages 31–37, July 2000.

[9] H. Röckle, G. Schimpf, and R. Weidinger. Process-Oriented Approach for Role Finding to Implement Role-Based Security Administration in a Large Industrial Organization. In *Proceedings of the Fifth ACM Workshop on Role-Based Access Control, Berlin, Germany*, pages 103–110, July 2000.

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

[11] R. Sandhu. Roles Versus Groups. In *Proceedings of the First ACM Workshop on Role-Based Access Control, Gaithersburg, Maryland, USA*, pages I–25–I–26, December 1995.

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

[13] D. Thomsen, R. O'Brien, and C. Payne. Napoleon Network Application Policy Enforcement. In *Proceedings of the Fourth ACM Workshop on Role-Based Access Control, Fairfax, Virginia, USA*, pages 145–152, October 1999.

[14] R. Witty and W. Malik. *Enterprise User Administration Magic Quadrant FY01, Research Note*. Gartner Group, January 2001.