Reconciling Role Based Management and Role Based Access Control

Emil Lupu and Morris Sloman

Imperial College, Department of Computing,
180 Queen's Gate, London SW7 2BZ, U.K.
E-mail: {e.c.lupu, m.sloman}@doc.ic.ac.uk

Abstract

Role Based Access Control is only a subset of the security management and distributed systems management. Yet, the characteristics and use of the role objects in RBAC or Role Based Management (RBM) may differ significantly. In this paper we outline a Role Management Framework based on the specification of policies and examine its differences and similarities with the RBAC concepts. In particular, two aspects of roles required in RBM are emphasised: the need for obligation policies which changes the way roles are used within the system and the Object Oriented role model which uses inheritance for re-use of the specification rather than implementing set-subset relationships on access rights.

Keywords

Distributed systems management, RBAC, role object model, role engineering.

1. Introduction

Role Based Access Control (RBAC) as presented in (RBAC 1995; Sandhu 1996) enriches the access control architecture with role objects which group a set of permissions. Users acquire these permission by being assigned to roles. The main objective of RBAC and its object model are to simplify the specification and management of *authorisation policy* – what actions subjects are permitted to perform on target objects. Access control is however only a subset of the security management functionality needed by a distributed system. There is also a need to specify and implement duties (*obligation policies*) which define actions to be performed by administrators or security components when events such as security violations are detected, e.g. the security administrator must investigate all sequences of 5 login failures from the same source or users must change passwords every 3 months. In this paper, we consider roles in the more general context of distributed systems management and show how these concepts can be used to extend the RBAC approach to cater for both specification and management of security.

Roles are originally an organisational concept which stem from the study of the behaviour of individuals. Each individual is assigned to a particular position in the organisation e.g. technical director, marketing manager for SW region, research assistant in the DSE section. A role represents the specification of the behaviour associated with a particular position in the organisational context (Biddle 1979). The main benefit of Role Based Management (RBM) is the ability to specify behaviour in terms of duties (obligations) and permissions (authorisations) by a consistent group of management policies – see section 2. A person or automated agent can then be separately assigned or removed from a role/position (Sloman 1994; Lupu 1997b) without having to re-specify policies. While not strictly part of RBAC, obligation policies are essential to security management since they specify the proactive and coercive actions needed for the management of security i.e. the *need to do* aspects. Both RBAC and RBM permit multiple users to be assigned to a role to indicate that they have the same permissions, but the duties related to an RBM role must be shared according to a predefined protocol. For example, only one of the users (whoever is free at the time) must perform an action related to an obligation.

The RBAC object oriented model (Sandhu 1996) organises roles in a set/subset hierarchy where a senior role inherits access permissions from more junior roles. While this approach of inheritance from instances optimises the use and definition of access rights at system level, it sacrifices the capability of parameterizing role instances upon creation from a pre-defined role class. We will show in (Section 3) why the ability to parameterise instances is

useful for security management and how such an object model can be built.

Only the very basic aspects of the role and policy framework, necessary for the understanding of the discussion, are presented here. For a more complete discussion refer to (Sloman 1994; Lupu 1995; Marriott 1996 a&b) and implementation of authorisation is described in (Lupu 1995; Yialelis 1996a; Yialelis 1996b).

2. Role based management

Roles are an important aspect of distributed systems management. As systems grow larger in size it is necessary to decentralise the management activities amongst multiple administrators and automated agents. In addition it must be possible to dynamically load and retract policies from agents to change the behaviour and strategy of the management system without re-coding or interrupting their activities (DSOM 1994; Sloman 1994; Koch 1996; Magee 1996). In this section we outline some aspects of the management policy notation and show how roles can be conceived as groups of policies.

2.1 Management Policies

Policies express a relationship between a domain of subjects (managers) and a domain of target managed objects. Domains are used to group objects in a hierarchical way similar to a file system (Sloman 1994b). They are more powerful than the grouping construct used in security systems in that they can contain nested domains and can be used to group both subjects and targets. A **User Representation Domain (URD)** is a persistent representation of a person within the computer system. It groups the adapter objects allowing the user to interact with the system (cf. login shell). Policies which apply to a user's *personal activities* within the system, independent of any roles, are specified in terms of the URD. Both positive and negative policies can be specified. Negative obligation policies can be seen as constraints or subject based filters which detect invocations outside the normal behaviour of the managers.

Authorisation policies define what activities a subject (manager or agent) can perform on a set of target objects e.g.

A+ *domain_administrators { "user_profile": modify();
remove(); reset() } *dse_domain
when (time< 08:00) && (time > 20:00)

Domain administrators are permitted to modify, remove or reset objects of type user profile in the dse domain outside office hours.

A- * students {workstation: reboot()} nt-pc

Members of the student domain are forbidden form rebooting the workstations in the nt-pc domain

Obligation policies define what activities a manager or agent must or must not perform on a set of target objects. Positive obligation policies are triggered by events and constraints can be specified to limit the applicability of the policy based on time or attributes of the objects to which the policy refers.

O+ on rlogin_event AC_agent { enable_encryption()}
*/applications/transfer_protocols

This positive obligation policy is triggered by remote login event and results in the Access Control (AC) agent enabling encryption on all transfer protocol objects.

O– x:*admin_agents { disable(); retract()} *policies
when x.state == standby

This negative obligation policy specifies that administrator agents must not disable or retract policies when they are in standby state even though they may be authorised to do so.

The **subject** of a policy specifies the human or automated managers and agents to which the policies apply and which interpret obligation policies. The **target** of a policy specifies the objects on which actions are to be performed. Security agents at a target's node interpret authorisation policies and manager agents in the subject domain interpret obligation policies. Both subject and target can be defined using a domain scope expression which identifies a set of objects in terms of union, difference, intersection and membership operators over sets of domains and objects e.g. *users - *administrators denotes all members of the users domain which are not members of the administrators domain. An advantage of specifying policy scope in terms of domains, is that objects can be added and removed from domains to which policies apply without having to change the policies. Assigning a permission to a user is therefore equivalent to including the user in the policy's subject domain. Similarly, extending the users' permissions to a new object can be done by extending the target scope of a policy to a new object. The **actions** e.g. enable(), retract() specify what must be performed for obligations and what is permitted for authorisations. Authorisation policies may optionally indicate object type for target domains containing objects of more than one type. Note that both

actions and targets have to be explicitly defined in a policy. We can therefore consider the case where two roles contain policies which authorise the same actions (access methods) on different target objects e.g. network administrator for the North, South and East sub-networks. These policies may be instantiated from a policy template (Section 3.1) which specifies the authorised actions and the constraints. This is not possible in RBAC where a permission has to define both the method and the target on which it is invoked. The **constraint** limits the applicability of a policy, e.g. to a particular time period, or making it valid after a particular date.

Policies are a general specification of authorisations and obligations of agents in the system. Although policies may be grouped in roles (as detailed in the following section), some policies have nothing to do with roles for example the policies applying to all members of an organisation or rights of an individual to their private files. RBAC often has to define complex pseudo roles relating to individuals to permit private ownership of files, also the fact that groups are sometimes used as a mechanism for implementing roles, has sometimes led to the merging of these two independent concepts.

2.2 A role as a group of policies

A role groups the policies specifying the duties and rights of a particular position inside the organisation. These policies reference a common subject domain called the **Manager Position Domain (MPD)**. A user is assigned to a role by authorising the user to connect to a proxy object in the MPD which inherits all the rights pertaining to the role and acts as the user's representative in that role (Figure 1). The user interacts with the system via an adapter object in the URD which is similar to an X server in that it provides a separate window for each role. This permits a clear separation of activity context for each role to which a user is assigned, and makes sure a user does not use the rights pertaining to one role to perform operations within another role. By analysing the policies referencing the URD it is possible to determine to which roles the user has been assigned. The main advantage of specifying policies in terms of roles rather than individuals is that organisational changes, when individuals are assigned to new roles, does not require any changes to the policy specification relating to the roles.

The sessions (connections in Figure 1) of the user are independent and the actions related to each session require specific authorisation policies. This enforces the principle of assigning permissions on a *need to do* basis and can be checked at specification level by a tool which may also detect other inconsistencies (Lupu 1997a). The complete separation of sessions is different from the RBAC framework presented in (Sandhu 1996) where users are allowed to combine the access permissions of several roles into one session. Conceptual consistency is therefore emphasised over flexibility and reuse.

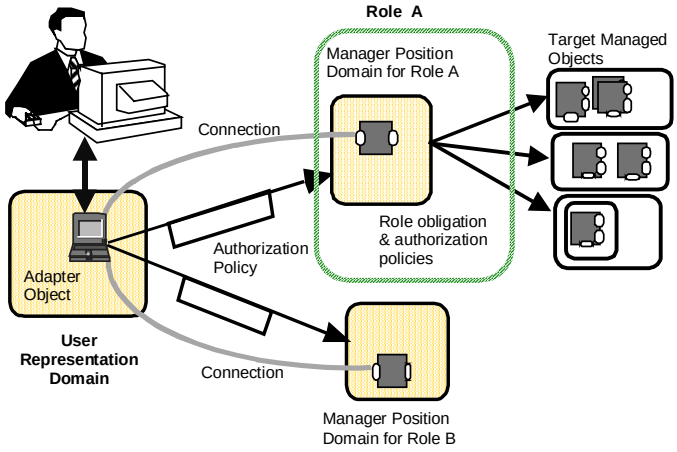

Figure 1 Management Roles

By analysing the policies of a role it is possible to make a per-subject "before the fact" review of the access rights (Gligor 1995). Moreover, the authorisation policies of the role indicate the permitted actions and can be used to customise the menus or choice of commands presented to the user in the window.

The use of obligation policies in a role differentiates the role from a group of users but also constrains the way roles are used within the system when multiple managers are assigned to a role. Consider the role of a nurse in a particular ward of a hospital. It may define the following policies in which the subject is a manager position domain (MPD):

/ the nurse is authorised to administer analgesics to patients */*

A+ MPD { administer(analgesics) } @/patients

/ the nurse must administer analgesics to the patient whose temperature exceeds 38°C */*

O+ on (x.temperature > 38**)** MPD { administer(analgesics) }
x:patient

If several users are assigned to the nurse role, the meaning of the authorisation policy is that all the users have permission to administer analgesics. However the obligation policies may have two different semantics: (1) all the subjects have to perform the activity e.g. all nurses must log their handling of drugs or (2) the activity must be performed by only one of the subjects. For example the obligation policy above expresses a responsibility or duty which is **shared** among the users assigned to that role so that only one of them administers the analgesics. In general, obligation policies which are part of a role are assumed to have a semantics of type (2) while an obligation policy applying to a group of users specifies that *all* subjects must perform the activity (semantics of type 1). The need to interpret the role as 'shared' between the users who are assigned to it does not arise in the RBAC framework (Sandhu 1996) since RBAC considers only authorisations and not obligations. This has implication on assigning multiple users to a role. For example, all engineers of a given project may be assigned to a single RBAC role since they have the same access rights, provided they can still be held individually accountable for their actions. Different roles are needed in RBM in order to reflect the different duties of developers or testing engineers, etc.

In the RBAC model (Sandhu 1996) special administrative roles, forming a separate hierarchy are needed to manage the roles and permissions. In our RBM framework, roles and policies are ordinary objects which can be included in domains so they can be the targets of other policies. Any role can thus be defined to manage other roles or policies. We also specify interaction protocols between roles (Lupu 1997b), but this is outside the scope of this paper.

In the following we examine some issues relating to the role object model and in particular the use of classes and their consequences in terms of access control.

3. Role Object Model

An organisation may contain large numbers of roles with few differences between them. It is thus desirable to facilitate reuse of both role specifications and permissions. It should be possible specify a role class which can be used to create multiple instances of that class. In addition, it can be useful to define a new role instance by specialisation (inheritance) of an existing role instance. However these two aspects of object-orientation do not combine well. The RBAC approach allows direct inheritance between instances e.g. in (Sandhu 1996) a role may inherit permissions directly from another, but the RBAC permissions include both the access method (i.e. read(), update(), enable()) and the target objects. Therefore, two roles which have the same authorised access methods but applying to different target objects are entirely distinct. In RBM however, it is possible to specify an authorisation policy template containing the access methods and the constraints and instantiate this template for different target objects. In order to achieve this it is necessary to introduce the definition of a *role class* from which instances can be created. For example a nurse-class role can be specified and used to create the nurse-instance roles for wards 3, 4 and 10. Specialisation inheritance is then used to define subclasses of role classes e.g. a surgical nurse role-class can be defined from a nurse-role class. In this section we explain the role object model and examine its uses. The definition of role classes is based upon policy templates (which are specifications of duties and rights independent of subject, target or both).

3.1 Policy Templates

Policy templates are used in order to provide the reuse of the policy specifications. A *policy template* may represent subjects and/or targets by symbolic variables. The template specifies the policy actions and constraints which can be reused for different subjects and targets. For example in a hospital a policy template such as the one below may be specified (S and T represent variables).

```
/* subjects are authorised to administer analgesics when
the temperature of the target is between 37 and 38.5 */

A+ S { administer(analgesics) } x:T when (x.temp > 37) &&
     (x.temp < 38.5)
```

The following policy authorising a nurse to administer analgesics to lung-disease patients may then be created from the above template by specifying the subject domain S and the target domain T.

```
A+ @/personnel/nurses { administer(analgesics) }
     x:@/patients/lung-diseases
     when (x.temperature > 37) && (x.temperature < 38.5)
```

A policy template may not inherit from another policy template since its components (actions, condition, trigger, etc.) are closely related and cannot be combined by inheritance. Note, that only the target of a policy template, which is part of a role class, has to be specified in the instantiation process since the subject is determined by the MPD. Policy templates are essentially a specification tool. Authorisation templates do not map onto access permissions, only the instances created from them do.

3.2 Role Classes

A role class groups specifications of the policy templates specifying the duties and rights of a generic role in the organisation e.g. nurse, engineer, marketing manager. When a role instance is created, a MPD for the role is then created so a manager cannot be assigned to a role class, only to a role instance. The role class contains only policy templates, not instances. Consider the example of the role class of a nurse, containing a set of policy templates which may have some undefined targets. The nurse class may contain the following templates:

/ the nurse is authorised to access the drugs database */*

pt_1 **A+** D { read(), search(), update() }
@/software/databases/drugs_db

/ nurses must monitor their patients */*

pt_2 **O+** D { monitor() } P

When the role is instantiated the MPD can be assigned to the variable D of template pt_1 which is then fully specified as the same target domain is used for all nurse instances. However pt_2 also needs a target domain representing the specific patients for which the nurse instance is responsible, to be assigned to variable P. Note, that instead of specifying the pt_1 policy template, a global policy instance can be defined which is not part of the role. This policy has a subject domain into which all nurses must be included which can be difficult to implement and to check.

In many organisations, there are roles which are essentially a variation of other roles with some additional rights and duties or some removed. We are experimenting with modelling this using both single and multiple inheritance. Figure 2, which uses the OMT style notation (Rumbaugh, 1991), shows a hospital scenario in which a surgical nurse role class inherits from a specialised nurse class, which in turn inherits from the nurse class. The specialised nurse redefines the policy pt_2 which it inherits. The paediatric nurse inherits from both the nurse and generic childcare classes. Multiple inheritance may result in more than one policy with the same name derived from the super classes. This can be resolved either by textual precedence in which policies from the first named superclass over-ride those from later classes or the programmer can explicitly indicate precedence.

A role class maintains a table of policy templates defined locally. Each entry has a name (the string representation of the name in the source code) and an object reference which is used by the support system to locate the object implementing the policy template. Note that policy templates and role classes are implemented as objects in our RBM framework and may be distributed on multiple servers in the system. The name of a policy template entry must be unique within the class scope and must not overlap with the name of an entry defined in the super-class(es). Adding an entry with the same name as in a super-class causes the policy template to be overridden.

There is an on-going debate within the object-oriented community as to whether a subclass should be a strict *superset* of the super-class. This results in a proliferation of intermediate classes holding common subsets of policies. For example, to accommodate specialised nurse and nurse having different versions of pt_2, it would be necessary to define a new class called generic nurse to hold all the policies common to these two classes, and then nurse and specialised nurse classes can each extend the inherited generic nurse class by defining an additional policy pt_2. This is the approach taken in the RBAC object mode, but we consider it inflexible and complicated.

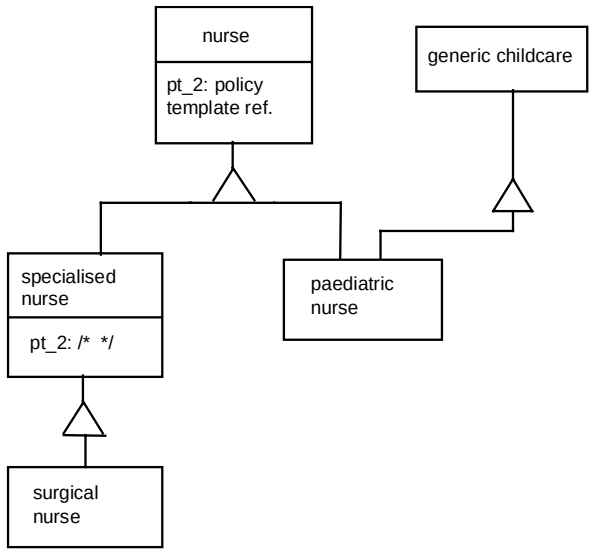

Figure 2 Role class inheritance graph

A class based approach to role engineering simplifies the effort of specifying the organisational structure by parameterising the creation of instances with the target objects to which the access rights are considered. For example two nurse for wards 3 and 4 may have permissions relating to the same operations on different target objects. The activities and rights of a nurse may be specified in a class and instances may then be created and customised from the common class specification.

3.3 Combining class inheritance and instance inheritance

Another issue that we are investigating is whether to permit specialisation of existing role instances, as in the RBAC object model. This can be very useful to model the situation where a new role has similar duties and rights over the *same target objects* as the one it is derived from. For example, in figure 3, the paediatric nurse and the paediatric surgery nurse have responsibility for the same patients in ward 3. It would be less work to specialise the ward 3 paediatric nurse instance than to derive a new paediatric surgery nurse role class from the paediatric nurse role class, create an instance from it and set all the target domains of the policies to ward 3. However it is not clear that supporting too many different forms of inheritance is a good thing as it makes the model more difficult to understand, and the support tools more complicated. Another problem, is that the surgical nurse may inherit redundant obligations from the paediatric nurse in ward 3 e.g. they may both end up with obligations to administer analgesics (see section 3). This may be an erroneous specification in that responsibility has been given to two people to perform the same task in the same circumstances. A solution would be to have a subdomain in ward 3 for patients who have recently undergone surgery for whom the surgical nurse has responsibility.

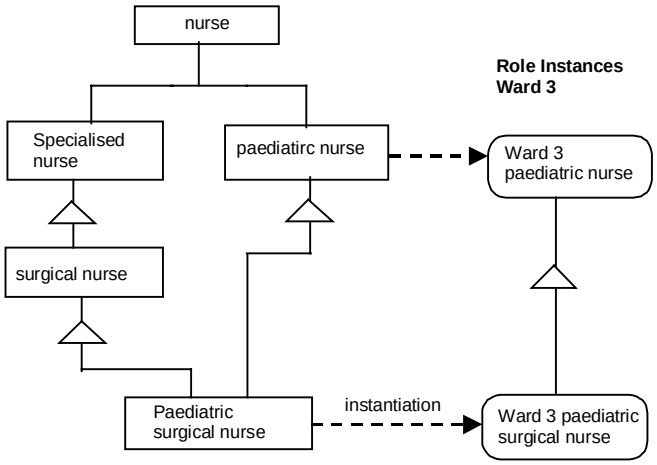

Figure 3 Instance and Class Inheritance

Our conclusion is that there is a definite requirement for class inheritance but we are not convinced of the need for instance inheritance. Further evaluation of the requirement for different roles with shared responsibility for the same targets is required.

4. Conclusions

Although RBAC is a subset of RBM, their role models are different. RBAC roles are an extension of existing access control mechanisms while RBM roles are derived from organisational roles and include duties as well as access rights. RBM roles are created from role classes rather than from other instances, to cater for the many similar roles in an organisation.

Obligation policies are essential for specifying security management actions to be performed periodically or triggered by events as well as the general duties related to the "need to do" aspects of roles. However there is a need to define a protocol for co-operation between multiple users assigned to a RBM role e.g. whether only one performs the obligation actions or all must perform the action. Our current approach is to assume only one object in the role performs actions but we may need a means of specifying other application specific cooperation protocols.

The definition of role classes in RBM and the parameterisation of instances with specific target objects is possible because our policies distinguish between the access method that is authorised and the target object on which the method is authorised. This does not seem to be feasible in RBAC since permissions include the target objects. The class based inheritance in the RBM framework provides reuse and specialisation of role specifications. The RBAC object inheritance model specifies a set/subset relationship between role instances and optimises the use of access rights. It is not clear whether both approaches are needed. We need to identify application requirements for multiple derived roles pertaining to the same target objects and to what extent duties differ between these roles.

Our current RBM framework consists of an editor for defining policy templates and instances plus tools for disseminating polices to access control or manager agents which interpret them. We have a prototype tool for specifying role classes and instances offering a graphical interface which hides the underlying entry/reference structure. In addition the tool caters for the definition of relationships between roles. We have also developed tools for detection of positive/negative conflicts between policies and detection of obligation policies specifying actions for which there is no explicit authorisation (Lupu1997a). We are currently experimenting with tools for checking the consistency of roles based on structural constraints e.g. separation of duties and concurrency constraints. These tools have been implemented using a CORBA based distributed programming environment.

5. Acknowledgements

We gratefully acknowledge financial support for the EPSRC RoleMan project (GR/K 37512), from British Telecom for the Management of Multimedia Networks Project and Fujitsu for the Policy based Telecommunication Management Project. We also acknowledge the contribution of our colleagues to the concepts described in this paper.

6. References

Biddle, B. J. (1979). "Role Theory, Expectations Identities and Behaviour", Academic Press Inc.

DSOM (1994). Proceedings of the IEEE/IFIP Distributed Systems operations and Management, Toulouse (France).

Gligor, V. (1995). "Characteristics of Role-Based Access Control". First ACM/NIST Workshop on Role Based Access Control, Gaithersburg, ACM Press.

Koch, T. and et al. (1996). "Policy Definition Language for Automated Management of Distributed Systems", IEEE 2nd. Int. Workshop on Systems Management, Toronto (Canada).

Lupu, E. C., et al. (1995). "A Policy Based Role Framework for Access Control", First ACM/NIST Role Based Access Control Workshop, Gaithersburg, ACM Press.

Lupu, E. C. and M. S. Sloman (1997a). "Conflict Analysis for Management Policies". IFIP International Symposium on Integrated Network Management (IM formerly known under the acronym ISINM), San Diego, Chapman & Hall publishing.

Lupu, E. C. and M. S. Sloman (1997b). "Towards a Role Based Framework for Distributed Systems Management", *Journal of Network and Systems Management*, **5**(1).

Magee, J. N., et al. (1996). Special Issue of *IEE/BCS/IOP Distributed Systems Engineering Journal* on Services for Managing Distributed Systems, **3**(2).

Marriott, D. and M. S. Sloman (1996a). "Implementation of a Management Agent for Interpreting Obligation Policy", IEEE/IFIP International Workshop on Distributed Systems Operations and Management, L'Aquila, Italy.

Marriott, D. and M. S. Sloman (1996b). "Management Policy Service for Distributed Systems", IEEE International Workshop on Services in Distributed and Networked Environments (SDNE 96), Macau.

Parker, T. and C. Sundt (1995). "Role Based Access Control In Real Systems", First ACM/NIST Workshop on Role Based Access Control, Gaithersburg, ACM Press.

RBAC (1995). Proceedings of the First ACM/NIST Workshop on Role Based Access Control, Gaithersburg, ACM Press.

Rumbaugh, J. et al. (1991) "Object Oriented Modelling and Design", Prentice-Hall International.

Sandhu, R. S., et al. (1996). "Role-Based Access Control Models", *IEEE Computer* **29**(2): 38-47.

Sloman, M. S. (1994). "Policy Driven Management for Distributed Systems", *Journal of Network and Systems Management* , **2**(4): 333-360.

Sloman, M. S. and K. P. Twidle (1994b) "Domains: A Framework for Structuring Management Policy" in *Network and Distributed Systems Management,* ed. Morris Sloman, Addison-Wesley, pp.433-453.

Yialelis, N., et al. (1996a). "Role-Based Security for Distributed Object Systems", IEEE Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (IEEE WET-ICE 96), Stanford, California.

Yialelis, N. (1996b). "Domain Based Security for Distributed Object Systems", Ph.D. Dissertation, Department of Computing, Imperial College, London.