(3) A Policy Based Role Framework for Access Control

Emil C. Lupu, Damian A. Marriott, Morris S. Sloman, and Nicholas Yialelis

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

Abstract

We outline a framework for specifying management roles which defines both authorisation and obligation policies for a particular management position. The policies define a relationship between a subject (manager) domain and a target domain in terms of activities permitted or forbidden, which must be or must not be performed. Policies grouped within a role refer to the same subject domain and propagate to the managers assigned to the roles. We cater for both human and automated managers and include interactions and concurrency constraints to specify aspects of the inter-role relationships in our framework. The paper presents the role based management framework and explains the concepts of policy based roles, then briefly describes the implementation of access control based on domain membership.

1.0 Introduction

ACM RBAC Workshop, MD, USA
© 1996 ACM 0-89791-759-6/95/0011 $3.50

Role theory identifies concepts such as roles and positions for enterprise modelling in order to specify organisational structure and activities for individuals in the organisation. In [BIDD79], a **Role** is defined as *"a collection of rights and duties"* and a **Position** describes a status within the organisation. The role specifies management actions that represent the behaviour or *dynamic* aspects of the position, which is essentially a *static* concept. A role thus identifies the authority, responsibility, functions and interactions associated with a position such as vice president, board director, security administrator, or operator responsible for reactor number three. We model *rights* as **authorisation policies** that specify what activities a subject is permitted (or forbidden) to perform on a set of target objects. *Duties* are modelled as **obligation policies** that specify what activities a subject[1] must or must not perform on a set of target objects. A role is the set of authorisation and obligation policies that have a particular manager position as a subject [SLOW94]. The advantage of using a position as the subject of the policies is that individuals can be assigned to or withdrawn from their positions without having to respecify policies. Although our role framework is aimed at supporting generalised management of distributed systems, the authorisation aspect of our roles corresponds to **Role-Based Access Control** (RBAC). Our framework can therefore be used as the basis for specifying and implementing RBAC.

[1] We use the term "subject" to refer to an object representing a user, human manager, or an automated agent that can initiate activites within the system.

As mentioned above, our main motivation for implementing roles in the management system was to simplify the specification of policies so that they do not have to be changed when managers are assigned to new positions. The responsibility for assignment of individuals to roles can be clearly separated from the responsibility for specification of the roles. In addition, a role provides a clear grouping of the policies for a position, which simplifies the analysis of the permissions and responsibilities assigned to that position. In a conventional access control approach, based on access control lists, determining the permissions assigned to a subject may require an in-depth search of all target objects in the system. The people assigned to roles do not work in isolation but interact and cooperate with other roles. We have therefore extended our role framework to permit the specification of interactions between roles reflecting the relationships between different roles that occur in an organisation. The overall motivation for implementing roles in a computer system, which supports people, is thus to simplify the specification of the system as well as to provide a clearly defined specification of the rights and duties related to the organisational structure.

In Section 2.0 of the paper we discuss the components of our role framework. Section 3.0 gives a brief description of the access control mechanisms needed to enforce the authorisation policies. Section 4.0 presents our conclusions and further work.

2.0 Components of a Management Role Framework

2.1 Domains

We use **domains** to group objects to which a common policy applies, to partition management responsibility or for the convenience of human managers [MOFF93]. Objects may be members of multiple domains, e.g., a user may be a member of two different departments in an organisation or a workstation may be a member of a security maintenance domain to reflect different management responsibilities. A subdomain is a domain object that is a member of a parent domain. Policies applying to a domain will, by default, propagate to subdomains and to the objects within them, although this propagation can optionally be disabled. Our concept of a domain is very similar to that of a directory in a typical hierarchical file system, and it performs a similar function to that of a group in access control, but domains can be used to group targets as well as subjects, as there is a need to specify policies with respect to both subject and target groups in large distributed systems.

The role framework caters for both human and automated managers, because the underlying system deals in the same way with an automated agent and an object representing a human. A **User Representation Domain (URD)** is a persistent representation of the human within the system. When a person logs in, an adapter object (c.f. login shell) is created within the URD to act as the interface process between the person and the computer system. A role manager position is represented by a **Manager Position Domain**. Assigning a manager to a position merely implies including the manager's URD in the position domain; role policies will propagate and apply to the adapter (see Figure 3-1, *Position Domains and Roles*) created within the URD. Multiple URDs may be included in a position domain to represent the

sharing of a position by a number of managers and a URD can be included in several position domains if the manager performs multiple roles [SLOW94]. The policies of the role relate to target objects or to target managers, reflecting that a manager may be responsible for managing subordinate managers.

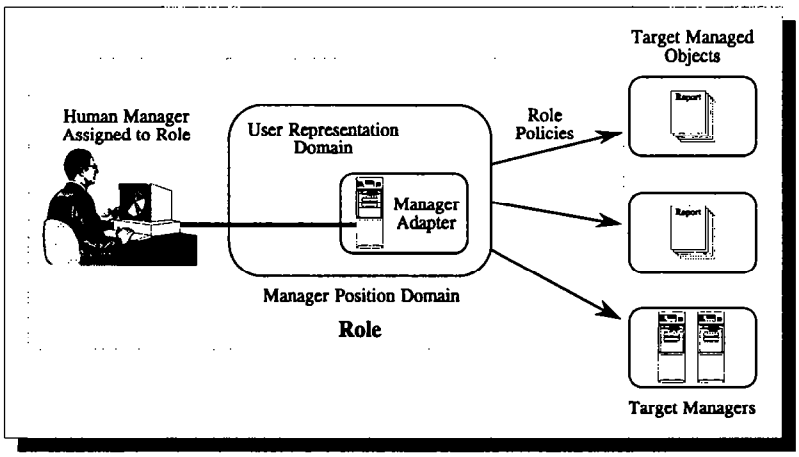

Figure 3-1. Position Domains and Roles

2.2 Management Policies

Policies define a relationship between a subject and a target domain. This relationship expresses either an **authorisation** (what activities the managers are permitted or forbidden to perform) or an **obligation** (what activities the managers must or must not perform on the managed objects). The **mode** of the policy distinguishes between positive authorisation (permitted: A+), negative authorisation (forbidden: A-), positive obligation (must: O+) and negative obligation (must not: O-). We permit the specification of high-level negative authorisation policies but our current system does not implement them in order to avoid conflicts (see Section 2.3). Obligation policies are triggered by events while authorisations are considered to be valid until revocation. Constraints limit the applicability of the policy e.g., between the hours of 09:00 and 17:00, and implement the concept of context based restrictions because they relate to particular attributes of the subjects and targets or to the state of the system. The general format of the policies is given below with optional arguments within brackets:

policy_id mode [*trigger*] *subject* { *action* } *target* [**when** *constraint*] ;

where 'subject' and 'target' denote sets of managers and target objects specified by domain scope expressions. The policy format and use is described in a recent research report [MARR95]. Example policies:

```
/* anonymous users are authorised to browse the Presentation Agent */
p_purchase_1 A+ u:users { browse() } Presentation_AG when u.type ==
    anonymous ;

/* on a connection request event, the security manager has to authenticate the
    user */
p_access_1 O+ on connection_request security_manager { authenticate() }
    u:users ;

/* anonymous users are forbidden to purchase */
p_purchase_2 A- anonusers { purchase() } Presentation_AG
```

Many negative authorisation policies can be refined into positive authorisations with constraints. For example, the above negative authorisation policy can be converted into a positive authorisation applying to users of type not anonymous, as indicated below.

```
/* non anonymous users are authorised to purchase */
p_purchase_3 A+ u:users { purchase() } Presentation_AG when u.type !=
    anonymous ;
```

Policies can specify actions at different levels of abstraction. A refinement hierarchy can therefore be built from the more abstract policies to the enactable leaf level policies (rules) [MARR95]. Abstract policies can only be interpreted by humans while leaf level policies are interpreted by automated components. The management policies, grouped in a role, scope the responsibilities relating to that role not only in terms of the activities to be performed, the target objects to which the activities relate but also in terms of the abstraction of the tasks to be performed. Policies can be changed, enabled or disabled dynamically to change the behaviour of the management system. Obligation policies are translated into a TCL script which is downloaded to the automated managers determined from the subject domain. Authorisation policies are given to access control components on nodes containing target objects for enforcement as described in Section 3.0.

An authorisation policy corresponds to "permission" in security terminology. Assigning a permission to a user is equivalent to including the user in the policy's subject domain. Similarly the policy can be made to apply to a new object or domain by including it within the policy's target domain. Authorisation policies could be implemented as either capabilities or as access control entries; we have chosen the latter. Policies are represented as objects and the policies pertaining to a role can be grouped into a domain. An authorisation policy can then specify which managers can modify or update the role policies. Another authorisation policy can specify the managers permitted to include or remove URDs from the role manager position domain, so it is easy to separately control the specification or updating of roles as well as the assignment of users to roles.

The domain service maintains information on the policies applying to a domain, so the access rights of a role can be determined from the authorisation policies applying to the role position domain. Similarly it is possible to determine who has access to a target object or domain from the policies applying to it, so either subject or target audits of the system can be achieved.

2.3 Extensions to the Role Framework

Large systems may have many roles which exhibit relationships such as client-server, supervisory-subordinate, contractual, collaborative peer-to-peer etc. Obligation and authorisation policies are insufficient to fully specify all aspects of role relationships and interactions. There is a need for an **interaction protocol** to define permitted interaction messages or invocation sequences which may be one-to-one or multi-party. There is also a need to specify **concurrency constraints** between the activities relating to one or more roles, i.e., sequencing and synchronisation of

the activities, e.g., activity A must be performed before B; activities C, D, and E can be performed in parallel but must all be completed before F is started. We provide a high level concurrency notation which is translated into compound events understood and enacted by a monitoring service [MANS95].

As represented in Figure 3-2, *General Purpose Management Roles*, our framework identifies for a role position: (i) the authorisation and

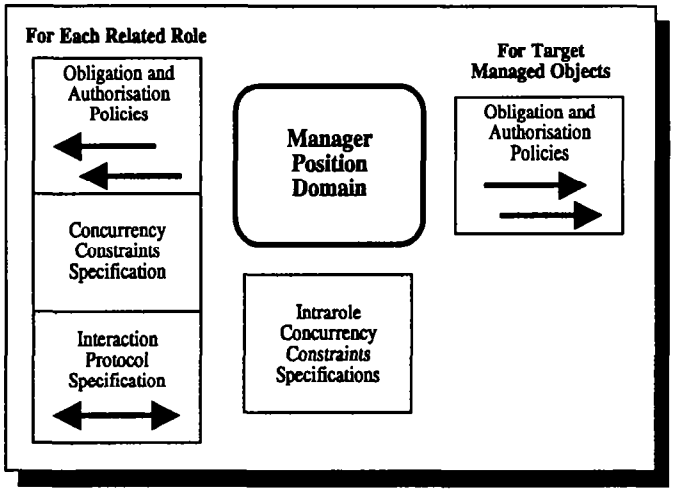

Figure 3-2. General Purpose Management Roles

obligation policies related to target objects which may be other managers or shared resources, (ii) the relationships between roles which reflect the organisational structure, and (iii) both intra- and inter-role concurrency constraints. This framework is aimed at general purpose management but can be used to specify RBAC which forms a subset of its full functionality.

Conflicts can occur between management policies and an appropriate conflict detection tool is needed to check the policies within a role or between related roles [MOFF94]. Modality conflicts potentially can arise from overlapping domains or if an obligation policy does not have a relevant authorisation policy to permit the activity (see Figure 3-3, *Conflicting Authorisation Policies*). It is not practical nor desirable to prevent such conflicts at the specification level in all cases as the only way to realise it would be by preventing domains from overlapping.

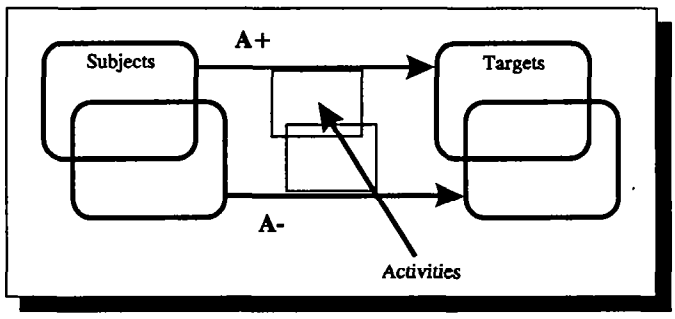

Figure 3-3. Conflicting Authorisation Policies

When such conflicts are allowed, criteria to establish the policy precedence have to be established. We have been experimenting with criteria based on domain nesting.

There are some application specific conflicts often arising from the principle of separation of duties, e.g., the person permitted to authorise payment for an invoice is not permitted to sign cheques. There is a need for specifying a **meta-policy**, i.e., policies about policies, which can help in the conflict detection and resolution regarding application specific conflicts. The meta-policies specify a logical predicate (we have experimented using Prolog for the implementation) and a domain of policies as the scope for the evaluation of the predicate. In terms of roles the meta-policies are constraints on the role structure describing acceptable role configurations [SAND96a]. The above mentioned example of the invoice payment can be written as:

any(P_1,P_2) belonging (domain_scope_expression)
    false <- intersect (P_1.managers, P_2.managers)
        && belongs('authorise', P_1.activities)
        && belongs('sign', P_2.activities)
        && belongs('cheque', intersect(P_1.targets, P_2.targets))

where P1 and P2 are the policies allowing a manager to authorise the payment of an invoice and to sign a payment cheque.

2.4 Role Classes and Inheritance

There has been considerable emphasis on the use of object oriented techniques for specifying roles [SAND96a, HU94a, NYAN94] but the use of inheritance to model the hierarchical structure of an organisation and the use of inheritance for reuse of role specification are sometimes confused. For example, a project supervisor may have responsibility over a test engineer and a programmer, but this does not necessarily imply that the project supervisor role should inherit the capabilities of the other two roles. However, in other cases it may be useful to define a new role class by inheritance from other role classes, e.g., a specialist physician has all the capabilities of a junior doctor plus additional ones. Object-orientation also permits the reuse of a role *class* specification by being able to create multiple role *instances* from it. For example, in a hospital there may be many nurses with a similar set of duties and permissions. Specifying a role class for nurses or doctors permits multiple instances of these roles to be created, but it must be possible to associate each role instance with a particular set of target objects reflecting the specific patients or resources for which each nurse or doctor has responsibility.

In our approach, creating multiple instances from a role class results in a position and target domains for each role instance, but with the same policies between position and target domains. These domains can then be populated with the relevant target objects to represent specific responsibility and a URD can be included in the position domain when a user is assigned to the role. It is also possible to define a policy class from which policy instances can be created. These concepts of class are analogous to contract templates where the contractual parties and the representative from the organisation enforcing the contract are not specified in the template but after instantiation.

3.0 Authorisation Policy Implementation

A security architecture is being developed which aims at enforcing authorisation policies and allowing the development of secure distributed applications on existing operating systems that do not support distributed security. A high degree of authentication and access control transparency is achieved by employing security agents on a per-host basis. These agents are trusted to act on behalf of the application objects on their hosts. Specifically, an **Authentication Agent** (AA) executes the authentication protocols on behalf of the application objects on the host. In addition, an **Access Control Agent** (ACA) holds copies of the authorisation policies applying to the objects on its host and determines whether a policy exists to permit a subject to access a target on its host. The Access Control List paradigm is combined with pseudo-capabilities which are used as hints to improve the time-efficiency of the access control decision mechanism. When a subject (which may act on behalf of a user) intends to invoke operations on a remote target, a **secure channel** is established between these two objects. The channel is identified by a unique *channel identifier* (chid) and is associated with cryptographic information (session key and cryptosystem used for secure communication), as well as access control information. The ACA on the subject host sends to the ACA on the target host the domain membership certificates and the pseudo capabilities (list of policies applying to the manager invoking the operation) which are used as hints to search the space of policies applying to the target object (Figure 3-4, *Propagation of Access Control Policies*).

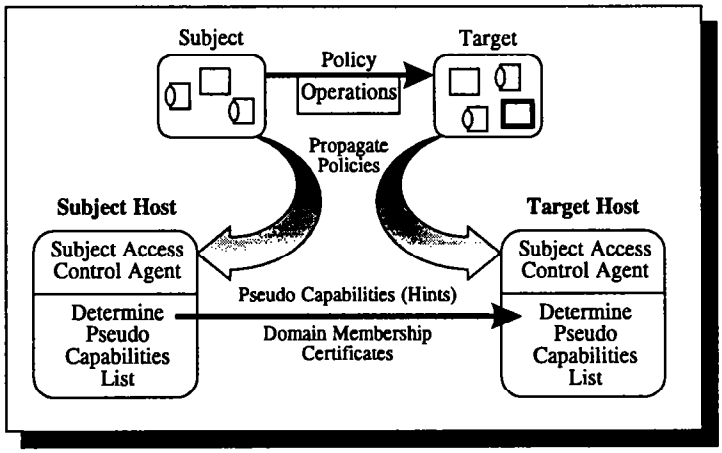

Figure 3-4. Propagation of Access Control Policies

The target Access Control Agent selects the policies permitting the subject of the established channel to access the target. The list of these policies is referred to as *Enabled Policy List* (EPL) and it is given to the *Reference Monitor* (RM) located in the address space of the target (Figure 3-5, *Access Control Overview*).

An RM makes access control decisions for target objects in its address space using the EPL that applies to a particular subject-target pair (identified by the *chid* of the established secure channel). Note that the determination of the EPL has to be done once for the lifetime of a

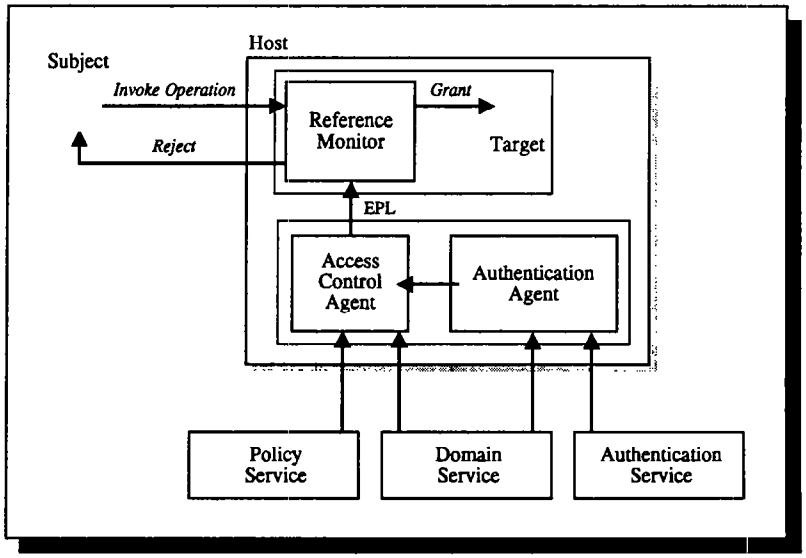

Figure 3-5. Access Control Overview

secure channel and is based upon the identity and domain membership of the subject. This is authenticated by the target *Authentication Agent* using a trusted *Authentication Service* based on symmetric cryptography. A detailed description of the authentication mechanism is given in [YIAL95]. When a new secure channel has been established, the session key for that channel is given to the cryptographic facilities in the address spaces of the subject and target objects. So, further communication between these two objects does not involve the authentication agents. A prototype of this architecture, detailed in [YIAL96], is being implemented using Orbix™.

4.0 Conclusions and Further Work

The model presented in this paper satisfies many of the requirements of a RBAC framework. Our domains permit grouping of both subjects and targets and help to cater for large scale systems in that policies apply to all objects in a domain.

Our authorisation policies provide a very flexible means of specifying access control permissions, and include constraints to limit the applicability of the policies. Because policies explicitly identify both subjects and targets, and domains maintain information about policies applying to them, it is easy to analyse the policies to determine those applying to a specific subject or target.

It is clear that roles provide a useful means of modelling organisational structure. However the inter-role duties, responsibilities and interactions need to be specified explicitly as in our role framework. Inheritance relationships between role classes indicate reuse of role specifications which may have nothing to do with hierarchical organisational structures, e.g., an engineer may possess access rights that a director cannot inherit as they pertain to activities requiring special skills.

™ Orbix is registered Trademark of Iona Technologies Ltd. Dublin, Ireland [IONA93].

Roles should not be seen as a means of solving all access control problems. A user assigned to a role should still have access to private resources such as diaries, email, personal files which pertain to that individual user and have nothing to do with their role. They will also wish to personalise their "desktop" to meet their own requirements, so it is not possible to fully define a user environment in terms of roles. Instead the role defines the additional specific applications and target objects accessible as part of the role. Applications can dynamically tailor themselves to specific roles, e.g., an application running on a user's workstation is given the authorisation policy for the user's role and only the menus, operations and target domains available for that role are made visible to the user. In our role framework, any policies (permissions) specified in terms of the URD are personal to the user and are not affected by the roles to which the URD is assigned, i.e., private target domains are still accessible. Access control decisions are made in terms of domain membership, which in the case of roles is the role manager position domain, and in the case of individuals is the URD.

The specification of the access control is drawn from the organisational policies which are often formulated in terms of what the users are not allowed to do, which corresponds to negative authorisations. Allowing both positive and negative authorisations within the same model, leads to potential inconsistent specifications which may be hard to detect and to resolve in an automatic way. Not allowing negative authorisations within the model does not completely solve the problem of inconsistencies. In particular conflicts related to the principle of separation of duty (i.e., the same person cannot be assigned to two particular roles) are application dependent and occur even if negative authorisations are banned. Our meta policies are used to define the constraints on permitted policies and the role structure to detect or prevent such conflicts. Another type of context dependent constraint, e.g., based on time, is used to limit the applicability of policies. The fact that there are multiple roles with managers executing activities in parallel result in the need for specifying concurrency constraints such as: activity a1 is authorised only if preceded by activity a2.

Our framework makes it easy to define a role administrator role who can create, delete and modify roles so the role structure is evolutionary and can be easily tailored to the needs of the organisation [REIN93]. Other manager roles may be permitted to assign or remove subordinate managers but not change the specification of the roles in the system.

In the current model, if the URD is included in multiple position domains, it inherits all the access rights from the various roles to which the manager is assigned. We are investigating the possibility of establishing connections between objects in the URD and adapter objects in the position domain. This would permit us to implement concepts such as "sessions" (see [SAND96a]). The current state of our work on roles is that tools exist for specifying policy hierarchies and we are evaluating techniques for specifying interaction protocols between roles, concurrency and meta-policies [LUPU95].

Other issues remain to be investigated including delegation and negotiated assignments. A very simple form of delegation of a role from Manager A to Manager B can be achieved by replacing A's URD in the

position domain by B's URD. In addition we are experimenting with extended authorisation policies which define what rights, with respect to a target domain, a subject can delegate to a delegatee to permit the delegatee to perform activities on behalf of the subject [YIAL95].

Acknowledgments

We acknowledge support from the EPSRC RoleMan Project, European Commission for the Esprit funded SysMan project (7026) and from Swiss Bank.

References

The Imperial College reports and papers are available on the Web from: http://www-dse.doc.ic.ac.uk or by FTP from dse.doc.ic.ac.uk.

[BIDD79] B. J. Biddle and E. J. Thomas, *Role Theory: Concepts and Research*. New York: Robert E. Krieger Publishing Company, 1979.

[HU94a] M. Hu, S. Demurjian, T. Ting, "The Factors that Influence Apropos Security Modeling and Analysis in the ADAM Object-Oriented Design Environment," *Database Security VIII: Status and Prospects, Proceedings IFIP WG11.3 Conference on Database Security*, J. Biskup et. al. (Eds), North Holland, 1994.

[IONA93] IONA Technologies Ltd., *Orbix™ - A Technical Overview*, Technical Report PN: PR-TEC-7-5, Dublin, Ireland, July 1993.

[LUPU95] E. C. Lupu and M. Sloman, *An Approach to Role Based Management for Distributed Systems*, Research Report DoC 95/9, London: Imperial College, July 1995.

[MANS95] M. Mansouri-Samani, *GEM - A Generalised Event Monitoring Language for Distributed Systems*, Research Report DoC 95/8, London: Imperial College, 1995. Available from ftp://dse.doc.ic.ac.uk/dse-papers/GEM.ps.Z.

[MARR95] D. A. Marriott, M. Sloman, and N. Yialelis, *Management Policy Service for Distributed Systems*, Research Report DoC 95/10, London: Imperial College, September 1995. Available from ftp://dse.doc.ic.ac.uk/dse-papers/maps.ps.Z.

[MOFF93] J. Moffett and M. Sloman, "User and Mechanism Views of Distributed System Management," *IEE/IOP/BCS Distributed Systems Engineering*, 1:1, August 1993, 37-47. Available from ftp://dse.doc.ic.ac.uk/dse-papers/2views.ps.Z.

[MOFF94] J. Moffett and M. Sloman, "Policy Conflict Analysis in Distributed Systems Management," *Ablex Publishing Journal of Organizational Computing*, 4:1, 1994, 1-22. Available from ftp://dse.doc.ic.ac.uk/dse-papers/conflict.ps.Z.

[NYAN94] M. Nyanchama, S. L. Osborn, "Access Rights Administration in Role-based Security Systems," Database *Security VIII: Status and Prospects: Proceedings of the IFIP WG11.3 Working Conference on Database Security*, J. Biskup, N. Morgenstern, and C. E. Landwehr (Eds), North Holland, 1994, 37-56.

[REIN93] G. L. Rein, B. Singh, and J. Knutson, "The Grand Challenge: Building Evolutionary Technologies," *26th Annual Hawaii International Conference on System Sciences*, vol. 4, Maui, HI, January 1993, 23-31.

[SAND96a] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman, "Role-Based Access Control," *IEEE Computer*, 29:2, February 1996, 38-47.

[SLOW94] M. Sloman, "Policy Driven Management for Distributed Systems," *Journal of Network and Systems Management*, 2:4, pp. 1994, 333-360. Available from ftp://dse.doc.ic.ac.uk/dse-papers/pdman.ps.Z.

[YIAL95] N. Yialelis and M. Sloman, *An Authentication Service Supporting Domain Based Access Control Policies*," Research Report DoC 95/13, London: Imperial College, September 1995.

[YIAL96] N. Yialelis and M. Sloman, "A Security Framework Supporting Domain Based Access Control in Distributed Systems," *IEEE Proceedings of the Internet Society Symposium on Network and Distributed Systems Security*," San Diego, CA, February 1996, 26-39.