Chapter 17

Specification of Management Policies and Discretionary Access Control

Jonathan D. Moffett

University of York,
Department of Computer Science,
Heslington, York, YO1 5DD, UK
Email: jdm@minster.york.ac.uk

This chapter is concerned with the specification of management policy. Its aim is to treat policies in a general manner, but discretionary access control policies are used as an example, to illustrate how they are specified and what the major issues are. It consists of four sections.

Section 17.1 gives the background and motivation for this subject and sets out a general model of management policies. It explains the need to model policies, their general characteristics, and their representation as objects so that they can be created, modified and queried. There are a number of possible relationships between policies, either hierarchical or potentially conflicting, and a brief analysis of this incompletely researched area is provided.

A well-understood example of a management policy is the access rule, used to specify discretionary access control policies. Section 17.2 introduces and explains access rules. A short example shows their practical effect, and the implementation issues are discussed.

In Section 17.3 delegation of authority, the means by which access rule administration can be carried out in a controlled manner, is introduced. The roles of Owner, Manager and Security Administrator in a typical organization are explained. The downwards flow of authority is modeled and controlled by fixed policies acting on

Management Role Objects (MROs). The use of MROs in ensuring the separation of powers for Security Administrators is explained.

17.1 Management policies

We are concerned with large distributed processing systems, which typically consist of multiple interconnected networks and span the computer systems belonging to a number of different organizations. Policies cannot be delegated or imposed from one central point, but have to be negotiated between independent managers.

Large distributed processing systems may contain hundreds of thousands of resources[1] and be used by thousands of users[2]. This implies that it is impractical to specify policies in terms of individual subjects or targets. We need to be able to specify them as relationships between groups of subjects and groups of objects. For example, the same policy may apply to all people in a department or to the set of files pertaining to an application.

All formal organizations have policies, with two related purposes: to define the goals of the organization; and to allocate the resources to achieve the goals. The policies are used to influence management, in a hierarchical fashion. A high-level policy guides a manager, who may achieve the policy goals by making lower-level policies which apply to other managers lower in the hierarchy.

Most organizations issue policy statements, intended to guide their members in particular circumstances. Policies may provide positive guidance about the goals of the organization and how they are to be achieved, or constraints limiting the way in which the goals are to be achieved. Other policy statements allocate (give access authorization to) the resources which are needed to carry out the goals. If they allocate money they are typically called budgets.

A common theme in distributed system management is the need for independent managers to be able to negotiate, establish, query and enforce policies that apply to a defined general set of situations.

An example of interaction between independent managers arises from the interconnection of two network management domains such as a Public Network (PN) and a local Imperial College (IC) network. This requires communication between the PN and IC network managers in order to exchange management information and establish access rules. Let us suppose that there are two relevant policies in force: PN policy gives the PN Manager the authority to carry out all relevant management operations on the network; and IC policy requires the IC Network Manager to report regularly on the status of the academic subset of PN nodes. We call these managers the **subjects** of the policies. In the absence of any other policies, the PN Manager has

---

1 Referred to as **target objects** or **targets** in this chapter.

2 In this chapter **user** refers to a human end user of a system. The representation of users by User Representation Domains is discussed in Chapter 16. Active entities in the system, representing or acting as agents for users, are referred to as **subjects.** Subjects act as target objects when operations are performed on them.

the authority to provide the regular status information, but no motivation to do so, while the IC Network Manager has the motivation to obtain the information but no authority to do so. The initial situation is shown in Figure 17.1(a).

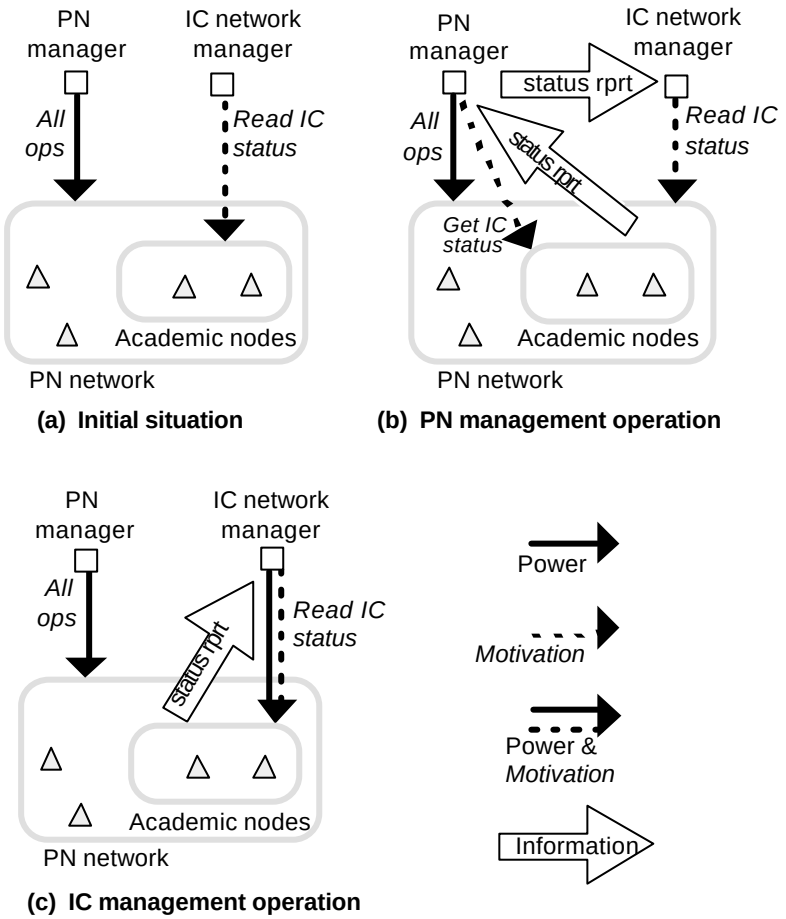

Figure 17.1 Policies for PN and IC managers.

An additional policy has to be established (created) by the PN Manager to meet IC's requirements. One approach is to create a policy that motivates the PN Manager to generate the status information and provide it to the IC Network Manager regularly, as shown in Figure 17.1(b). An alternative approach is to create a policy that gives the IC Network Manager the authority to perform the operations needed to obtain the regular status information, as shown in Figure 17.1(c).

This example brings out one of the main points in the model. Policies that motivate activities and policies giving authority to carry out activities can each exist independently of the other. However, if only one of the two kinds of policies exists in

relation to an action, the action will not be performed. For management activities to be carried out, there needs to be a manager who is the subject of both kinds of policy: a policy giving authority to carry out the activity; and a policy motivating the manager to do so.

17.1.1 The need to model policies

The example above shows that there is a need for a means by which independent managers can query, negotiate, set up and change policies. It can of course be done by the well-tried method of telephone calls and the exchange of paper, but there are potential benefits in using the distributed system itself to communicate and store policies, particularly with respect to automated management. There is thus a need to be able to represent and manipulate policies within a computer system. It is important that the representation of policies and the protocols used to negotiate them should be uniform across management applications.

Storing an organization's policies in a database permits staff to search, using keywords, for policies relevant to their proposed plans. The Pythagoras project (Bedford-Roberts, 1991) was concerned with modeling policies in order to create a database of the policies of an organization. Its purpose was for users to query the database, matching textual patterns in the policy statements, in order to enable them to ascertain what policies may exist in a specified subject area. The system does not interpret or constrain the contents of policies.

With the automation of many aspects of management in distributed systems and computer networks, there is the need to represent management policies within the computer system so that they can be interpreted by automated managers in order to influence their activities. One specific class of policies, which we term **management action policies**, is particularly useful when considering distributed system management, and is discussed in detail in Sections 17.1.2–17.1.4. Other classes of policy are briefly considered in Section 17.1.5.

17.1.2 Characteristics of management action policies

We define some characteristics of the policies that we will be discussing in order to give a working definition that is more precise than simply "plans". We start from the basis that policies are intended to influence actions. However, policies are not concerned with instant decisions to perform an action, instantly carried out. If a manager specifies that something is to be done once only, and instantly, he or she does not create a policy, but simply causes the action to be carried out. Whether the policy defines a single future action to be carried out, or repeated actions, or relates to the maintenance of a condition, it needs to have *persistence*.

Policy modalities – motivation and authorization

As shown in the example above, we distinguish between policies that are intended to motivate actions to take place and policies that give or withhold power for actions to take place. **Actions** are operations that are performed by agents provided two preconditions are satisfied: motivation and power (see Figure 17.2):

- **Motivation** is the term we use to imply that the agent wishes to carry out an action, and will do so provided he or she has the power to do so.
- **Power** implies that if an agent attempts to carry out an action, he or she will succeed.

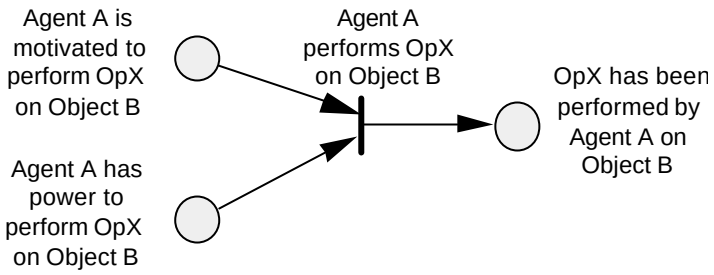

Figure 17.2 Preconditions for action.

One method of acquiring power is through delegated **authority**, which we define to be legitimately acquired power. The very concept of a policy implies a well-ordered world, and so policies are rarely concerned with unauthorized power. We take the simplifying view that all policies giving power can be viewed as giving authority, and in the rest of this chapter we categorize policies as being either authorization or motivation policies.

Agents are always humans, but it is convenient to extend the concept of "action" to include computer processes which have been commanded to carry out actions. An agent's motivation is represented by the agent submitting a command to the computer system. An agent's authorization is represented by more than one means: for an action to be carried out successfully the access control system must authorize the agent's use of named resources such as files, while his or her use of commodity resources such as file store and CPU time must also be authorized by the system, perhaps through an accounting application.

17.1.3 Policy attributes

Policies, whether concerned with motivation or with authorization, have at least the following attributes: Modality; Policy Subjects; Policy Target Objects; Policy Go als; and Policy Constraints.

Modality

A policy has one of the following modalities: **positive authorization** (permitting), **negative authorization** (prohibiting), **positive motivation** (requiring), and **negative motivation** (deterring). We do not exclude the possibility of other useful policy modalities being proposed, but these are adequate for the present analysis.

Policy subjects and target objects

Policies are about organizational goals, which need someone to achieve them. All policies in this model have subjects, the people to whom they are directed, that is, those who are authorized or motivated to carry out the policy goal within the limits defined by the policy constraints. The Policy Subjects attribute defines a set of subjects. Where a policy has been automated as a computer system command we regard the user who will input the system command as the policy subject. The Policy Target Object attribute defines the set of objects at which the policy is directed.

The sets of policy subjects and target objects may both be specified either by enumeration or by means of a predicate to be satisfied. Individual policy subjects and target objects are not normally specified, because a policy is typically expressed in terms of organizational positions and domains of objects, not individuals. One approach to specifying organizational positions and enumerating groups of objects is by using generic **management domains**; see Chapter 16.

Policy goals

The Policy Goals attribute may define either **high-level goals** or **actions**. An example of a high-level goal is "recover from media failure"; it does not prescribe the actions in detail, and we can imagine a number of different actions that could achieve the goal. On the other hand, an example of an action is "run the BackUp job on department D's disk files". Note that the distinction between high-level goals and actions may depend upon the context. In an environment in which there is one standard "back up" operation defined, a goal such as "back up department D's disk files" might be regarded as an action policy, while in other situations it might be a high-level goal in which the System Administrator has several options for action.

In order to distinguish between actions and goals, we need to have an alphabet of the operations that can be performed by objects in a system. Then any goal that is expressed purely in terms of the operations is an action, and any other goal is to be regarded as a high-level goal. **Procedures**, often found in organizations' policy manuals, can be regarded as a sequence of actions.

Actions represented by operations naturally have three components: the target object on which it is performed, the operation performed, and one or more parameters to the operation. Policies may relate not only to the operation name, but also to parameters of the operation. In the example, "Financial managers are authorized to carry out financial transactions of a value up to £1 million", the amount of the transaction is a parameter of the operation, and forms a part of the authorization policy.

Policy constraints

The Policy Constraints attribute of a policy object places constraints on its applicability. They are predicates which may be expressed in terms of general system properties, such as extent or duration, or some other condition. An example of constraints in authorization policies expressed by access rules is the limits on the terminal from which the operation may be performed, and/or limits on date or time, for

example “Members of Payroll may Read Payroll Master files, from terminals in the Payroll office, between 9 a.m. and 5 p.m., Monday to Friday”.

17.1.4 Representing management action policies as objects

It is useful to view management action policies as objects on which operations can be performed. For simplicity we assume the following minimal set of operations:

- Create a policy
- Destroy a policy
- Query a policy

Authorization may be required to perform operations on policy objects. If the computer system is simply a documentation aid, no restrictions may be needed. On the other hand, if the policies are actually used to influence system actions, as in the case of access control policies, restrictions on operations are required. They are discussed in detail for authorization policies in (Moffett and Sloman, 1991).

Although we regard all policies as objects, some may be altered dynamically while others are **fixed policies**, fixed for the life of the system. The following example is implicitly an example of a fixed policy: “The system coding is to ensure that it is impossible for a user to be logged on at two terminals simultaneously”.

The object that expresses a fixed policy will typically be separate from the coding that implements it. On the other hand the system could use this policy object as part of its implementation; for example, the policy could be defined by an object with a Read-only attribute stating the number of terminals at which the user could be logged on, which the system queries when appropriate. The reason for recording it as an explicit policy object will be to ensure that it is recognized as deliberate, and not removed as an undesirable restriction at the next software release. Using it as part of the implementation will enable systems with differing policies to be generated more easily.

17.1.5 Relationships between management action policies

The subject of the relationships between management action policies is a complex one, which has not yet been researched completely. At this stage it is possible to identify two main sorts of relationship: policy hierarchies; and overlaps which may result in conflicts. It is also possible to throw light on the concept of responsibility by modeling it as two related policies.

Policy hierarchies

The organizations using and managing distributed processing systems are hierarchical in nature and authority is delegated downwards from senior management. It is a fundamental characteristic of policies that they are organized in a hierarchical fashion. The following example is illustrated in Figure 17.3:

(a) The Managing Director of a company is to protect its assets from loss.

(b) The Manager of Department D is to protect all files from loss owing to fire or media failure.

(c) The System Administrator is to back up Department D's disk files weekly.

(d) There is a system command (which was input by the System Administrator) to run a job which backs up Department D's disk files to tape each Thursday at 22.00. The System Administrator is to take the tapes to a safe store in a different building each Friday morning.

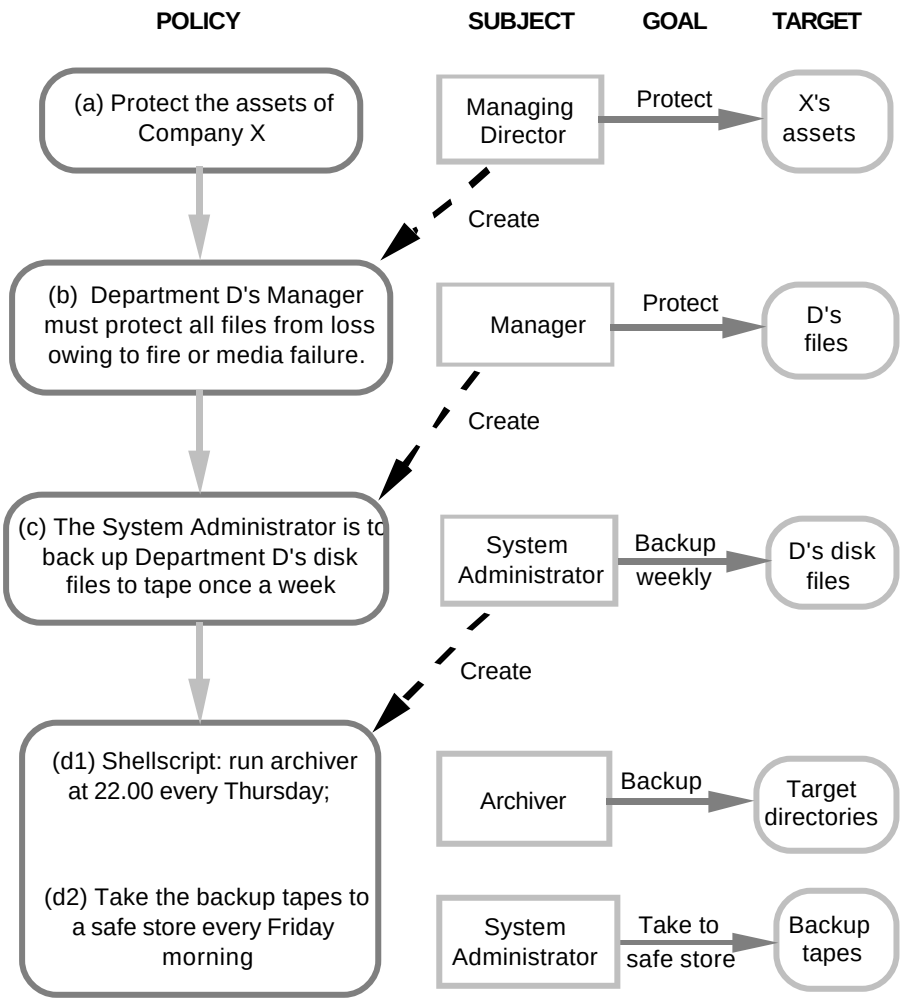

Figure 17.3 A policy hierarchy.

We may suppose that the board of a company has made policy (a) and given responsibility for carrying out the policy to the Managing Director. The Managing Director has made policy (b) and given responsibility for it to the Manager of

Department D. The Manager has made policy (c) and given responsibility to the System Administrator, who has carried out that resposibility by creating policy (d). This is actually a procedure consisting of two actions, one automated and one manual.

Policies are made at a high level by senior management and responsibility for achieving their goal is assigned to other members of staff, who may in turn do one of the following:

i) Create a lower-level policy that will achieve the policy goal they have been assigned, and assign responsibility for it to another member of staff. This is what was done in policy (b).

ii) Delegate the task not to another member of staff, but to a computer system that will carry out actions that achieve the goal. This is what the System Administrator has done in the first part of procedure (d).

iii) Achieve the specified goal themselves by carrying out actions that achieve the goal, which the System Administrator does in the second part of procedure (d).

Policy conflicts

The policy model has opened up the possibility of systematic analysis of policy conflicts. This analysis is at an early stage, but a first step has been taken by recognizing that the overlap of objects – either policy subjects or target objects or both – between policies is a necessary condition for conflict. If there are no objects in common between two policies, there is no possibility of conflict.

Conflicts of modalities can be recognized independently of any specific application. They assume overlaps between the subjects, goals and target objects of the two policies:

- There may be a direct contradiction of modalities, for example two simultaneous authorization policies, one (positive) permitting and one (negative) prohibiting the same subject to perform the same action on the same object. Some means of preventing or resolving conflicts of this kind is essential for any system, and typically this is achieved by giving policies a priority ranking.
- There may be an inconsistency between motivation and authorization. When a subject is motivated to perform an operation on an object but not authorized to do so, the organization in which this occurs will not be able to achieve its goals, and the conflict will need to be resolved.

There are also policy conflicts that are not inherent in the model, but are dependent upon the application. Several different modes of application-defined conflict emerge from the analysis of the ways in which policies may overlap:

- *Subject and target objects both overlap* – where the subject and target objects of two policies both overlap, there may be a conflict of operations, for example "The same person may not both Input and Approve a payment transaction".
- *Target objects overlap* – when the target objects of two policies overlap, there is a potential conflict arising from multiple managers of a single object. This

may result in one manager reversing the operation carried out by the other, unless a coordination mechanism is introduced.

- *Subject objects overlap* – when the subject objects of two policies overlap, there is potential conflict of interests, such as the same person having management authority over two competing organizations.
- *Subject and target objects overlap in one policy* – when subject and target objects overlap in one policy, there is the situation of a manager managing himself. This is again a potential conflict which is application-dependent; it may be acceptable for an automated manager to configure itself, but not for a human manager to sign his or her own expenses.

Responsibility

The concept of responsibility is frequently associated with policies. It can be analyzed in our model in terms of the relationship between two motivation policies, for example: the Head of Department decides that the System Administrator shall be responsible to the Administration Manager for backing up the department's files. This can be modeled as two related motivation policies, each created by the Head of Department:

- One policy motivates the System Administrator to do the backup actions, and report to the Administration Manager;
- A second policy motivates the Administration Manager to supervise the System Administrator.

This analysis enables two different concerns to be separated out: responsibility *for* and responsibility *to*, which are often confused in informal discussion.

Policy analysis using deontic logic

Deontic logic, the logic of normative systems, which enables reasoning about **obligation**, corresponding roughly to motivation, and **permission**, corresponding to authorization, is potentially a useful tool in policy analysis. Conflicts of modalities have been discussed in the context of deontic logic by Alchourron (1991), among others. The analysis of the concept of responsibility into two separate policies follows Kanger (1972) who makes the distinction between responsibility *for* and responsibility *to*.

17.1.6 Other classes of policy

Although management action policies are useful for modeling several policy applications, not all policies fit easily into this framework. We discuss briefly two other classes of policy – domain membership constraints and policies about policies.

Domain membership constraints

Management domains were introduced in Chapter 16. Some policies may be expressed as constraints upon domain membership, for example:

- There must be at least two members of the domain of Security Administrators.
- Processors in a domain must be able to support the M68000 instruction set.
- Destroying an object is prohibited if the object is still a member of another domain.

There are of course two possible kinds of constraint. The first, which we favor, is a constraint upon the operations that affect domain membership, typically the create and destroy operations on any object and the include and remove operations on a domain. These add and subtract objects to and from the policy set. The predicate defined by the constraint has to be evaluated once only, when the operation is performed. Constraints on the number or type of objects in a domain, and on the values of read-only attributes, can be enforced by this means. The second kind of constraint is a general predicate, which the system is required to maintain, about the attributes of members of a domain. This is potentially very difficult to achieve, because every time any application functional operation attempts to change an object's attribute, the system is required to verify that the domain constraints, of every domain of which the object is a member, are not violated. We do not therefore envisage constraints of this kind being permitted.

Policies about policies

Some policies apply to management action policies rather than to other system objects. An example is the policies, discussed in Section 17.3, that constrain the operations that can be performed on access rules. Access rules are themselves management action policy objects, and it is necessary to place quite complex restrictions on how they can be created, altered and destroyed. These restrictions cannot themselves be expressed easily as management action policies, and we make no attempt here to model them systematically. However, there is clearly an important research task in doing so, as it is necessary to be able to reason about the relationships between all kinds of policies that may affect the actions performed upon objects.

17.2 Discretionary access control policies

Many aspects of policy modeling are still in their infancy. However, security policies have been researched quite thoroughly and we use, as an example, the management action policies for discretionary access control which are known as **access rules**.

Access control policy relates to authority, that is, power which has been legitimately obtained, and to how authority is delegated. Access control is concerned with ensuring that subjects and processes in a computer system gain access to computer-based resources in a controlled and authorized manner. Resources such as files must be protected from unauthorized access and access permission must be assigned only by subjects or managers with authority to do so. We consider only *logical* access control and not issues relating to *physical* access control such as locks and secure rooms. The USA Department of Defense Trusted Computer System

Evaluation Criteria (DoD, 1985) define two kinds of logical access control. **Discretionary access control** permits managers or other subjects to specify and control the sharing of resources with other subjects. **Mandatory access control** enforces policies that are built into the design of the system and cannot be altered except by installing a new version of the system. For example, in multi-level security systems, data cannot be read by a subject with a lower security classification than has been assigned to the data. Mandatory policy is particularly applicable to military environments, whereas discretionary policy typically applies to commercial, industrial and educational environments. This chapter is concerned with the latter.

We make the assumption that there is no inherent right of access of any kind for ordinary subjects. If a person is not the owner of an object, and has not been given authority, then the computer system should refuse all access.

In general, authority is given not to a person but to a position or role within an organization. A typical policy is that "the Payroll Clerk is authorized to change the Payroll Master file". While John occupies the position of Payroll Clerk he has that authority, but when he is replaced in that position by Mary, he loses the authority and Mary gains it. An important aspect of access authority is the ability of senior management to delegate it down through an organization. We have identified distinct management roles which apply to commercial organizations, each with different authority, and in Section 17.3 we introduce the concept of **delegation of authority**, as a means of the controlled dissemination of access authority within an organization.

We use the **reference monitor** concept, as described in (Anderson, 1972 and DoD, 1985), to model an access control system. The function of a reference monitor is to enforce the authorized access relationships between subjects and targets of a system. It is a trusted component of the system. All operations requested to be carried out on an object are intercepted by the reference monitor, which only invokes the operation if the access is authorized. In this case access control is transparent to the subject, but otherwise the subject is informed by a failure message. In general, access control is carried out by the system, and not by the target object itself, although the reference monitor could be implemented by the object.

17.2.1 Access rules

Access rules were introduced in (Moffett *et al.*, 1990). They are the means for specifying access control policy and are an adaptation of Lampson's **access matrix** (Lampson, 1974) for large distributed systems. The access matrix is unsuitable for specifying policy in large distributed systems because it assumes global knowledge of the objects in the system, and specifies policy in terms of individual objects rather than groups. Access rules overcome these problems by being localized and by specifying policy in terms of sets of objects, using management domains.

An access rule object, as introduced in Chapter 16, has the attributes **subject domain**, **target domain**, **operation set** and **constraints**. It can be seen that this corresponds exactly to a policy object whose modality is positive authorization, in which the policy subjects and target objects are the objects in the subject and target domains, and the policy goals are the operation set.

Both subject and target domains can be the names of domains or more general domain expressions. The operation set defines the names of the operations, defined in the object's interface, which the access rule authorizes. There is no assumption that one operation "implies" another, for example "Write" permission implying "Read" permission. The constraints limit the applicability of the access rule.

The system authorizes a request if an access rule that matches the operation request exists. An access rule matches an operation request if the subject[3] attempting the operation is in the subject domain of the rule, the target object is in the target domain of the rule, the operation name is in the operation set of the rule, and the conditions specified in the constraints are met. See Figure 17.4. If no access rule that applies to the operation request exists, then authority for the operation does not exist and access is denied.

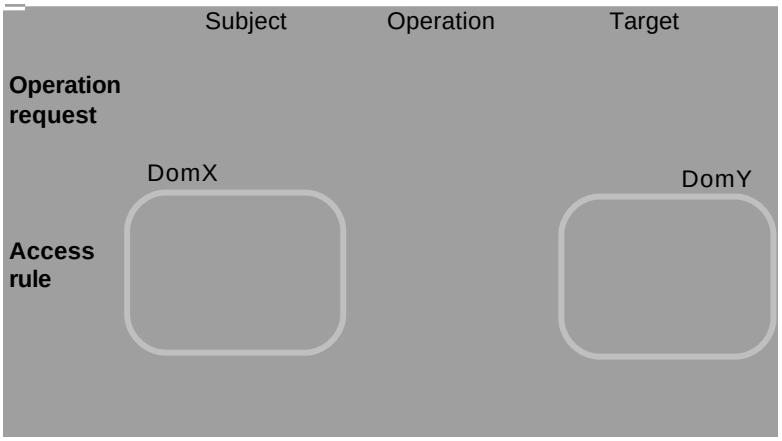

Figure 17.4 An operation request and a matching access rule.

It is possible that there are overlapping access rules – more than one access rule satisfies the requirement. No conceptual difficulty arises from this, as it indicates that authority has been granted via more than one route, but there may be practical difficulties when attempting to remove a subject's access rights; these are discussed in Section 17.2.3, below.

There is a need to modify access rules dynamically during the life of the system. We model this by treating access rules themselves as objects which are instances of an access rule type. Access permissions are given and removed by creating, destroying and modifying access rule objects. The need to control the granting and removal of permissions is achieved by imposing rules for the creation, destruction and modification of access rule objects. This is discussed in Section 17.3, Delegation of authority.

---

[3] We assume that the system has authenticated the identity of the user.

Domain expressions

The purpose of the Subject and Target Domain fields of an access rule is to enable the specification of sets of objects to which it applies. Domains and subdomains are a powerful means of doing this, but are limited in what they can achieve. Therefore the Subject and Target Domain fields are specified in terms of domain expressions which allow either a single domain name, or a more complicated expression, to be specified, in particular the standard operators on sets, **Set Difference** and **Set Intersection**.

If domain DomA contains DomB and DomC, then an access rule applying to DomA applies to the union of the members of DomB and DomC. However, it provides no means of expressing other basic set operations such as Set Difference and Set Intersection. Set Difference can express groups of objects such as "all subjects in Payroll Dept. except the Payroll Clerks", and "all files in Payroll_Files except Payroll_Master". Set Intersection can express groups such as "all files that are in Payroll_Files and also in Personal_Data".

Domain manipulation operations could be used to create a new domain with the required membership, but in order to ensure that set operations in access rules are always evaluated at the time the rule is checked, Set Difference and Set Intersection operators are allowed in domain expressions. There is also a notation which allows the specification in a domain expression that the access rule should apply only to direct members, and not to members of subdomains also, which is the normal case.

Constraints in access rules

The constraints in an access rule permit its applicability to be limited in terms of time of day, date, and location of user. For example, it is quite common in many organizations to permit access to various objects only during normal office hours or to set up an access rule that expires after some time or only becomes valid in the future. Access to some resources may only be permitted for a particular group of terminals; for example, access to Accounts Payable data may be permitted only from the terminals in the Payments Office of an organization.

We assume that the Reference Monitor has available information about the user's environment (date and time of request, and location of user), and the parameters of the operation.

Logging switch in access rules

A Security Audit Facility requires logging of both authorized and unauthorized operation requests. It is assumed that the Reference Monitor will provide it on all unauthorized operations, but there may also be a selective need for information on authorized operations also. For example, all changes to access rules performed by a security administrator should be logged. One means of controlling this would be by a logging switch in access rules, which could be turned On in order to trigger the Reference Monitor to provide the Security Audit Facility with input. Adequate controls are required to prevent unauthorized switching Off.

17.2.2 Example of access rules

We use as an example the Payroll Department of an organization. It consists of roles for people, and a directory of payroll files; see Figure 17.5. We assume that the Payroll Manager (viewed as being outside the department he runs and not shown in the figure) can create whatever access rules he wishes.

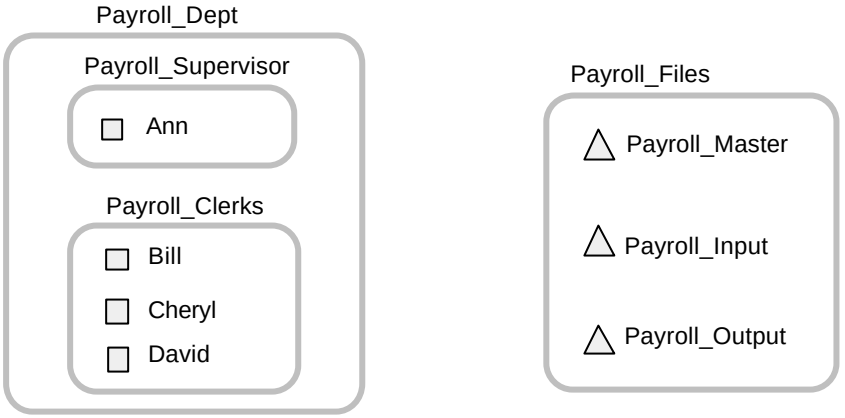

Figure 17.5 Domain structure of the payroll department.

The Payroll Manager may not know or understand the names of the files in the Payroll_Files domain or the names of the files that will be added to it in future. However, he has delegated to the Payroll Supervisor the task of maintaining the files, so an expression of this delegation policy is to give the Payroll Supervisor the ability to Create, Read and Write files in that domain. His policy is also to allow the whole Payroll Department to Read the Payroll_Files. These policies are expressed by the two access rules illustrated in Figure 17.6.

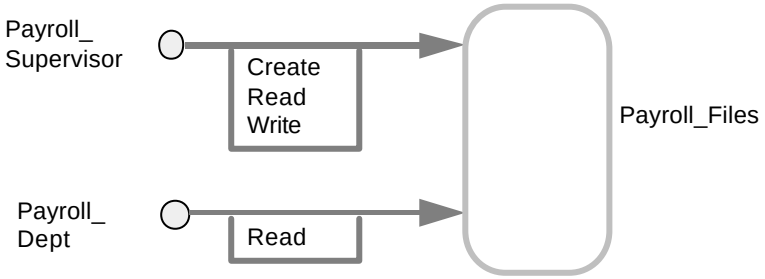

Figure 17.6 Example access rules.

The access rules do not name the subject or target objects, so we cannot tell directly from them whether access by a particular subject to a particular target will be allowed. To interpret the access rule we have to know the domain structure, as shown in Figure 17.5, in order to identify the subjects and target objects to which it applies.

17.2.3 Removal of access

The prompt removal of access authority when necessary is an essential aspect of access control. There are a number of methods of achieving this:

i) It is simple to express the exclusion of a particular subject from accessing a target domain at the time an access rule is set up by means of a domain expression such as (Subjects_X – Subjects_Y), which excludes members of Subjects_Y from access which they would have been permitted as members of Subjects_X. Note, however, that this will not override another access rule which allows members of domain Subjects_Y to have access. Similarly, a Target domain expression may specify (Files_X – Files_Y).

ii) Negative rights, which override any positive rights, have been used in some systems as a means of rapidly and selectively revoking access to sensitive objects (Satyanarayanan, 1989). This greatly complicates the semantics of an access control system and can introduce inherent contradictions (Tygar and Wing, 1987). We therefore conclude that they should not be introduced into our model and so we only permit positive authorization.

iii) Destruction or modification of an access rule will not prevent access if there is another remaining rule that permits access. Determining which access rules permit access can be complicated because typically the subject or target object is a member of several domains which are subdomains of those specified in access rules. Tools and reporting facilities are therefore needed to permit a security administrator to determine what target domains a subject can access and what subjects can access a particular target domain. The use of these tools would be adequate for routine removal of access rules, but searching a large system for relevant access rules may be rather slow.

iv) Denying access to an individual subject in an emergency should be performed not by alteration of access rules but by suspension or deregistration of the subject. It is instant and effective and does not add complications to the system.

There are thus a number of different strategies that can be applied, depending on the circumstances.

17.2.4 Implementation issues

The above section has described access rules as a means of specifying access control policy. This is the user view of security policy, which must then be reflected in an underlying implementation that makes use of access control mechanisms to implement the policy.

The main implementation issues are:

- Granularity of access rules – what is the finest level of granularity at which a user can specify domain expressions?

- How to store access rules – should they be associated with the subjects, target objects or independent of both?
- How to implement the reference monitor in a distributed system?
- How to translate from the user view (access rule) to the implementation mechanism being used?

Mechanism requirements

It is essential that the underlying implementation should make access control decisions which are consistent with the user view of domain-based access rules. When an access rule is created, modified or destroyed, the effect of the change must be reflected in access control decisions. It may be permissible in some circumstances for the effect to be delayed (for example, until the following day, in the case of non-urgent changes), but this must be predictable. In addition, when an object is included in or removed from a domain, or a new object is created, the object's access authorizations (either as subject or target) which derive from membership of the domain must change accordingly.

The performance cost introduced for validation of authorized operation requests must be minimal if the system is not to degrade. The overhead in updating access authorizations must be tolerable for domain membership changes. It is thought that the contents of subject domains will change relatively infrequently compared to those of target domains which contain objects such as temporary files. Changing access control policy by updating access rules, or obtaining status reports, will be relatively infrequent and does not always have to take immediate effect, so higher overheads can be tolerated.

Granularity of access rules

The finer-grained the unit of specification of objects in access rules, the more detailed the access control policy can be, but the greater the problem of implementing the mechanism with acceptable performance. The two main options, both for subjects and for target objects, is to set the finest grain either at a domain or at individual objects. The Andrew project (Satyanarayanan, 1989) allows individual subjects, as well as domains, to be specified in their Access Control Lists (ACLs), but the finest granularity for the ACLs of file objects is the Directory. The problem can be simplified by representing individual human users by a specialized domain, a User Representation Domain (URD), as described in Chapter 16.

Reference monitor implementation

When considering implementation issues it is useful to refine the reference monitor into two components – an Access Control Decision Facility (ADF) and an Access Control Enforcement Facility (AEF). The ADF compares an operation request against the current access control policies and makes a decision on whether the request is to be accepted or rejected. The AEF enforces the decision made by the ADF, and we view its job as ensuring that no operation message reaches the target object unless it has been authorized by the ADF. We therefore assume that the AEF is closely

associated with the target object, certainly by being in the same physical machine, and possibly by acting as a front end to the object.

This is superficially similar to the OSI Access Control Framework (ISO 10181-3, 1991). It should be noted, however, that the OSI Access Control Framework is restricted in its scope; it covers only the framework for access control *enforcement*, and not for specification of access control policy.

The ADF and access rules can be implemented as an independent authorization service, or held with the subjects or target objects.

- *Authorization service:* An access rules database can be independent of both subject and target objects as part of an authorization service. It would have to be based on a distributed database and distributed authorization servers. There is therefore a potentially heavy performance penalty associated with it.
- *Access rules held with subjects*: Access rules can be held as attributes of subject domain objects. The access rules (and ADFs) would thus be distributed to reflect the distribution of the subject domains and the ADF could be co-located with the domain and generate capabilities to be checked at the target object's node. The problem with holding access rules with the subject domains is that they may exist in personal workstations which cannot be trusted, and their security is therefore difficult to enforce. In addition, access rules are more likely to be generated by security administrators in the organization of a target domain rather than a subject domain.
- *Access Control Lists (ACLs) with target domains*: An alternative approach is to hold access rules as ACLs which are attributes of the target domains. The security of the access rules can then be commensurate with the security of the objects they are protecting. Target objects are more likely to be based on servers, under the control of system managers rather than ordinary users, and their domains and access rules can be held on more secure systems.

The target object and its domain may be on different computers. If an access control decision takes place only when a session is established between a subject and target object, then the overheads of a remote access by the AEF to the ADF co-located with the target's domain would be acceptable. However, if access checking is required on every operation invocation then remote access to the ADF would be unacceptable and every target would have to hold its own ACL.

The disadvantage of a pure ACL approach is that it can lead to a breach of security in systems with nested operations. If a server object has to invoke operations on other objects on behalf of a subject it usually adopts the identity of the subject in order to obtain subject-specific authorization for the operation. It has been pointed out by Vinter (1988) (among others) that this breaches the "least privilege principle" as the server can now perform all the operations authorized for the subject, and not only the one that was requested. This only works when servers are trusted, which may not be the case if the server belongs to another organization on a remote network.

17.2.5 Domino implementation approach

The Domino project (Law *et al*., 1990) uses the third approach, an implementation based on ACLs stored with domain objects (Twidle and Sloman, 1988). The testbed is the Conic toolkit for building distributed systems (Magee *et al*., 1989). The access control decision is made when an interface of one object is bound to the interface of another. Each object holds information on the domains of which it is a member. When a binding takes place the list of parent domains of the subject is passed in the binding request sent to the domain of the target object. It searches its ACL for a matching rule. If none is found there will be a search upwards through the domain structure for an entry in a parent domain's ACL that will authorize the request.

User and mechanism views of access rules

The distinction between the view of the user managing a system and the underlying mechanism(s) used within the system for implementing this view was introduced in Chapter 16.

Table 17.1 User and mechanism views of access rules.

User View		Mechanism View
Access rule object	<-------->	ACLs attached to target domains
Subject domain (name-based domain expression) Object set Constraints	<-------->	Combined to form an ACL entry which is an attribute of every domain affected by the target domain expression
Target domain (name based domain expression)	<-------->	Every domain affected by the target domain expression holds an ACL entry.

In Domino the user view of access rules is an access rule object, while the mechanism view uses ACLs as described above. The relationships between the user and mechanism views of access rules are illustrated in Table 17.1. Both views are implemented; the user, typically a security administrator, specifies policy by creating and destroying access rule objects, and the system alters the ACLs to reflect these policy decisions.

There is clearly a problem of ensuring consistency between the two views of access rules. Ideally, at the design stage, we should have taken our formal specification of the user view (Moffett, 1991) and either refined it to the mechanism view while ensuring the preservation of properties, or created a formal specification of the mechanism view and proved that the properties were preserved. However, in the absence of tools to support these activities we have opted for an informal design of the mechanism view.

There is also a problem of ensuring consistency between views during operation, as the two views can become inconsistent through system errors and

failures. Consistency checking and restoration tools are being implemented in order to diagnose and recover from any inconsistencies.

17.3 Delegation of authority

The model for delegation of authority must reflect the organizational management structure and policy as well as provide mechanisms for the transfer of authority from one agent to another. A resource owner should be able to delegate authority over his resources to another subject, within the mandatory constraints of the system. In addition it must be possible to delegate *the authority to delegate*, but limit the scope of this second-level delegation. For example, an owner of an organization should be able to delegate authority to a security administrator to give subjects access to objects in a defined part of it. Finally, the model should permit subjects to have personal domains where they can control the objects they create.

17.3.1 Management roles

A number of management concepts underlie our approach to authority and its delegation.

Ownership

It is assumed that humans are ultimately responsible for the actions of the system. In many situations they will use automated agents to perform operations within the system, but they will retain responsibility for the actions of the agents, and therefore require the power to control the agents. Thus we will always be able to trace responsibility back to human users, who we call the **owners** of the system (for example, managing director of a company or board of directors). Ownership in this chapter is intended to denote a concept as close as possible to normal legal ownership of goods and property. All computer systems in a country such as the UK process resources with identifiable legal owners who have legal powers over them. We regard ownership as the starting point for delegation of authority.

As a very rough approximation, ownership of an object implies the legal power to perform any feasible operation on it, together with responsibility for the operations that are performed. There are of course limitations to this power; for example, an owner of personal data may not disclose it except under the terms of data protection legislation, and the owner of petroleum spirit at a bulk distribution plant must ensure that an automated system dispensing it to tanker lorries does so in accordance with certain safety regulations. Restrictions of this kind have to be represented in computer systems as mandatory constraints, and are beyond the scope of this chapter.

Many previous authors, such as Lampson (1974), have assumed that the subject who creates an object automatically becomes its owner. We do not hold that view, but distinguish between ownership of objects and the delegated power to create them. A data processing clerk who submits a job to create a new version of a file of

bank accounts is not the owner of those accounts, but is carrying out the task of creating the file as an agent of the owner of the bank.

For the purpose of discussing authority we assume that we start with an owner, who can then dispose of or share ownership or delegate a subset of his or her powers to another person. Provided that this is done legitimately we will describe this other person as having gained authority. In a computer system authority is normally represented as the ability to perform defined operations on objects through specified interfaces.

Separation of responsibilities

Separation of responsibilities is an important control concept which is familiar in the context of auditing; see, for example (Waldron, 1978). It is designed to ensure that no one has excessive authority. Clark and Wilson (1987) have pointed out that the concept should be modeled in computer systems. It requires that different aspects of certain transactions should be carried out by different users, so that no one person can carry out the transaction autonomously. An example is the authorization of payment of suppliers' invoices, where the input of invoices to a computer system must normally be carried out by a different user from the person who can trigger off the actual release of payments. Neither can carry out the other's function, so the payments cannot be made without their cooperative activity. Requirements for separation of responsibility have to be specified at an application level but support for it needs to be provided by the access control system.

This separation of authority manifests itself in the role of a security administrator, who often is responsible for granting access authority in large organizations. The security administrator should not be allowed to grant himself access to the resources under his control.

Management structures

Each organization has its own management style, which is reflected by different management structures and policies for delegation of authority. For this chapter we have identified four typical roles within an organization management structure – User, Security Administrator, Manager and Owner. Owners can share ownership with other subjects, or can delegate Manager authority. Managers can delegate authority to Security Administrators, who can create access rules which allow ordinary subjects to perform operations on target objects. Note that it is an application decision whether a subject in a management role has authority over himself, for example a Security Administrator being able to give himself access rights. We specifically allow for this case to be prevented, but we do not enforce it as part of the model.

17.3.2 Delegation of authority using management role objects

When authority is delegated, there is a need to control to whom it is delegated, the resources over which the authority applies, and how the authority can be passed on down a management chain. The existence of an access rule allowing SubjectA to create an access rule object is clearly not a sufficient control by itself. It provides no constraint on the contents of the access rule object which is created, so SubjectA

would be able to create access rule objects to enable any subject to access any objects. We therefore need a means of placing controls on the contents of access rule objects.

A security administrator must be able to grant access authority which he does not possess himself, so the concepts of *having* access and *giving* access must be decoupled. Similarly owners may delegate, to managers, the authority to create security administrators. We therefore need to distinguish between the normal operations that a user can carry out, and operations that give authority.

17.3.3 Management role objects

As explained in Chapter 16, management positions can conveniently be represented by domains. However, many roles in an organization are associated with particular types and extents of authority. The concept of a management role object (MRO) extends the simple domain object in order to define the range and nature of the managerial authority of the members of a domain. Informally, it defines a type of role, which defines the nature of the authority, the subjects who occupy the role, and the scope of their authority. It is modeled as an object with three attributes:

- **Role Type**, the name of this type of role, which determines the structure of the Role Scopes attribute;
- **Role Subjects**, a domain expression defining a set of subjects who occupy the role;
- One or two **Role Scopes**, each a pair of (**authority type**, **domain expression**). Each domain expression defines the objects for which the role subjects have the corresponding type of authority.

In the typical management structure we are using here, there are role types of Owner, Manager and Security Administrator. The Owner and Manager role types each only have a single authority type. A Security Administrator's authority requires that the Security Administrator (SA) role type has two separate types of authority: SA Subject and SA Target, one to define the subjects and the other to define the target objects, for which he or she can make access rules. This is to achieve the separation of responsibilities for security administrators which is discussed below.

There may be multiple instances of MROs of a single type, such as a number of Managers each managing a different department. Occupation of a management role gives a subject the authority associated with that role. If a subject occupies more than one role he or she has the authority associated with each one; for example, someone may be the manager of more than one department in an organization. A subject's role in the organization changes if he or she is moved to another MRO.

We now discuss our sample management structure using MROs as the means of representation. Normal users of the system have no inherent ability at all to grant access authority, unless they are allowed to administer their own personal domains (see below).

Owner

An Owner has a set of objects (subjects and/or target objects) over which he has authority, defined by the scope of Owner authority of the role he occupies. He can give away or share ownership (create joint ownership), and delegate the Manager authority over these objects to other subjects. He does this by creating Manager MROs, with appropriate Role Subjects and scope of Manager authority. The limits of the Manager authority of an MRO that an Owner can create are set by requiring his Owner authority to be a superset of the Manager authority he is creating.

We remarked above that the creator of an object is not necessarily the owner of it. This follows automatically from our model; if a subject creates a new object in a domain, the ownership of that object, like all others in the domain, remains with the domain's owner. Indeed, the subject cannot even read the object after creation unless an access rule also allows him the Read operation.

Ownership is shared if more than one subject has Owner authority over an object. At this stage we take a simplistic view of transfer of ownership, by specifying that an Owner should be able to remove Owner authority from another Owner of the same object. This allows both for suicide and for destruction races where the first person to remove the other's ownership wins power. Further constraints on sharing and transfer of ownership will need to be considered.

Manager

A Manager similarly has a set of objects defined by the Manager authority of the role he occupies, over which he has authority. He can appoint Security Administrators and set the scope of their authority by creating SA MROs with them as Role Subjects, and appropriate scopes of SA Subject authority and SA Target authority. The limits of the SA Subject authority and SA Target authority of an MRO that a Manager can create are set by requiring his Manager authority to be a superset of the SA Subject authority and SA Target authority he is creating.

Security Administrator (SA)

A Security Administrator has two sets of objects over which he has authority, defined by the scopes of SA Subject authority and SA Target authority of the role he occupies. He can give authority to subjects to perform operations on objects by creating access rules, by virtue of his occupancy of a Security Administrator role. The limits of the access authority that a Security Administrator can create are set by requiring his SA Subject authority and SA Target authority to be supersets of the Subject Domain and Target Domain of the access rule he is creating. In our example system, no limit is placed on the operations he can permit, but a restriction of this kind could easily be added.

One of our main aims is to allow a Security Administrator to create access rules for other subjects while preventing him from giving himself access. This is done by ensuring that the membership of the Role Subjects and of the scope of the SA Subject authority of an SA MRO do not overlap. Then since the Security Administrator (an occupant of the Role Subjects of the SA MRO) cannot himself be a member of the scope of the SA Subject authority he cannot grant access authority to himself. This

achieves the desired separation of responsibilities between granting access and having access.

There is of course the need for the Security Administrator himself to have access to some objects, but he cannot create the requisite access rules himself. This need is met by a second security administrator who can allocate access to the first.

We illustrate the Security Administrator's power with an example. There is a large company which employs computer security administrators (members of the SA domain) part of whose job is to set the access rules in accordance with company policy. Members of the SA domain are given the authority to make access rules relating to the subjects and files in Dept_A domain. They are given similar authority for Dept_B domain. Their powers are illustrated in Figure 17.7. Note that they are

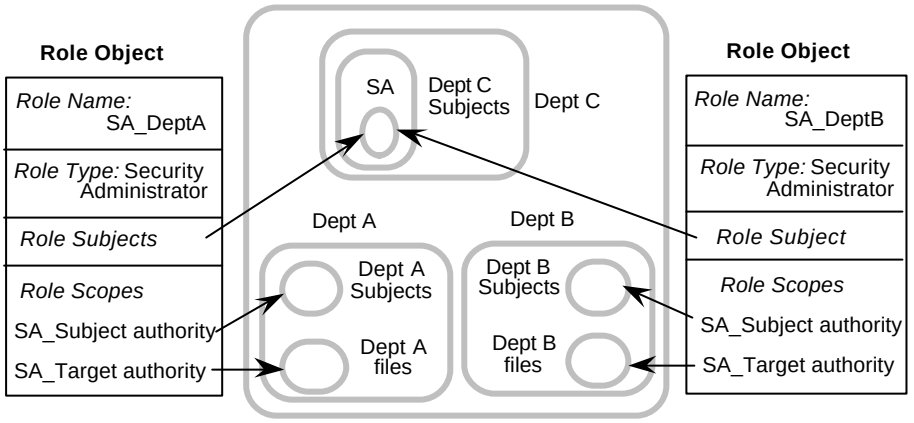

Figure 17.7 Security administrator roles for departments A and B.

not authorized to make access rules that allow subjects in Dept_A to access files in Dept_B, or vice versa. SA is a member of Dept_C, and therefore its members cannot make access rules to give themselves access.

Personal domains

A registered user of the system will have ext ensive control over a personal domain.. This can be done within this model by making each user a Security Administrator for his or her personal domain, typically with the ability to share access to that personal domain with anyone in the organization. An MRO is created for each user, with the user as sole Role Subject, SA Subject authority over all users, and SA Target authority over the personal domain. This achieves the objective, as illustrated in Figure 17.8.

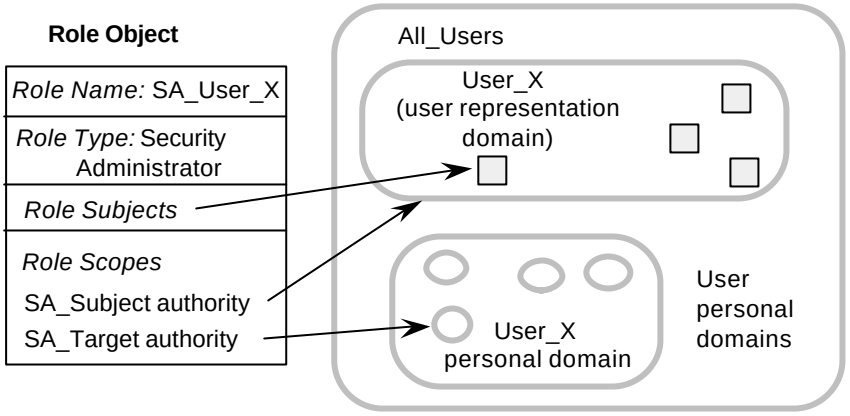

Figure 17.8 A subject acting as security administrator for his or her personal domain.

17.4 Summary

This chapter has shown how general distributed system management policies may be modeled, using discretionary access control as the main example.

Management policies are used to define the aims of an organization and to allocate the resources that are needed. They need to be modeled as objects, in order to enable the system to give support to the managers who implement them, with the aim of automating the activity where this is possible and beneficial. It is then possible to explore the relationships between policies in, for example, policy hierarchies and policy conflicts.

Discretionary access control policies, modeled as access rules, were taken as a well-developed example. Their use for security administration was illustrated. Delegation of authority was introduced as the means by which access rule administration can be carried out in a controlled manner. The downwards flow of authority from Owners through Managers to Security Administrators in a typical organization, ensuring the separation of powers for Security Administrators, was explained.

Abbreviations

ACL	Access Control Lists
ADF	Access Control Decision Facility
AEF	Access Control Enforcement Facility
MRO	Management Role Objects
SA	Security Administrator
URD	User Representation Domain

References

Alchourron C.E. (1991). Philosophical foundations of deontic logic and its practical applications in computational contexts, *Proc. First Int.Workshop on Deontic Logic in Computer Science (DEON'91)*, Amsterdam, The Netherlands, 11–13 December 1991

Anderson J.P. (1972). *Computer Security Technology Planning Study*, ESD-TR-73-51 vol. 1, AD-758 206, ESD/AFSC Hanscom, AFB Bedford, Mass, October

Bedford-Roberts J. (1991). Concepts from Pythagoras. *Report HPL-91-22*, February 1991, Hewlett-Packard Laboratories, Bristol, UK

Clark D.C. and Wilson D. R. (1987). A comparison of commercial and military computer security policies, *Proc. IEEE Security and Privacy Symposium*, 184–94

DoD (1985). *Department of Defense Trusted Computer System, Evaluation Criteria*, Department of Defense (USA) document DOD 5200.78 – STD, December

ISO 10181-3 (1991). *ISO Security Framework 3: Access Control*. ISO/IEC JTC1/SC21

Kanger S. (1972). Law and logic. *Theoria*, **38**, 105–32

Lampson B.W. (1974). Protection. *ACM Operating System Review*, **8**(1). 18–24

Law A.D., Sloman M.S. and Moffett J.D. (1990). The "Domino" Project. *Proc. Data Management '90 Conference*, Egham, UK, 2–3 April 1990, BCS Data Management Specialist Group, 143–54

Magee J., Kramer J. and Sloman M.S. (1989). Constructing distributed systems in Conic, *IEEE Transactions on Software Engineering*, **15**(6), 563–75

Moffett J.D. Sloman M.S. and Twidle K.P. (1990). Specifying discretionary access control policy for distributed systems, *Computer Communications*, **13**(9). 571–80

Moffett J.D. and Sloman M.S. (1991) Delegation of authority. In *Integrated Network Management II*. (Krishnan I. and Zimmer W., eds.), 595–606, North Holland

Moffett J.D. (1991). *Delegation of Authority for Access – A Formal Model*. Domino paper A2/IC/4.1 (revised May 1991), Dept of Computing, Imperial College, University of London

Satyanarayanan M. (1989). Integrating security in a large distributed system, *ACM Transactions on Computer Systems*, **7**(3), 247–80

Twidle K. and Sloman M.S. (1988). Domain based configuration and name management, for distributed systems, *Proc. IEEE Distributed Computing Systems Workshop*, Hong Kong, September 1988

Tygar, J.D. and Wing J.M. (1987). Visual specification of security constraints, *Proc. IEEE Workshop on Visual Languages*, Linkoping, Sweden, August, 288–301

Vinter S.T. (1988). Extended discretionary access controls, *Proc. IEEE Symposium on Security and Privacy*, April 1988, Oakland, CA, IEEE Computer Society Press, 39–49

Waldron R.S. (1978). *Practical Auditing*, London: HFL (Publishers) Ltd.