Journal of Network and Systems Management,
Plenum Press, Vol. 2, No. 4, 1994, pp. 333-360

POLICY DRIVEN MANAGEMENT FOR DISTRIBUTED SYSTEMS

Morris Sloman

Imperial College
Department of Computing
180 Queen's Gate, London SW7 2BZ
Email: m.sloman@doc.ic.ac.uk

Abstract

Separating management policy from the automated managers which interpret the policy facilitates the dynamic change of behaviour of a distributed management system. This permits it to adapt to evolutionary changes in the system being managed and to new application requirements. Changing the behaviour of automated managers can be achieved by changing the policy without have to reimplement them – this permits the reuse of the managers in different environments. It is also useful to have a clear specification of the policy applying to human managers in an enterprise.

This paper describes the work on policy which has come out of two related ESPRIT funded projects, SysMan and IDSM. Two classes of policy are elaborated – authorisation policies define what a manager is permitted to do and obligation policy define what a manager must do. Policies are specified as objects which define a relationship between subjects (managers) and targets (managed objects). Domains are used to group the objects to which a policy applies. Policy objects also have attributes specifying the action to be performed and constraints limiting the applicability of the policy. We show how a number of example policies can be modelled using these objects and briefly mention issues relating to policy hierarchy and conflicts between overlapping policies.

Keywords

Distributed systems management, network management, management policy, security policy, policy conflicts, access rules, domains.

1 Introduction

Distributed systems management[1] involves monitoring the activity of a system, making management decisions and performing control actions to modify the behaviour of the system. Policies are one aspect of information which influences the behaviour of objects within the system. **Authorisation policies** define what an manager is *permitted or not permitted* to do. They constrain the information made available to managers and the operations they are permitted to perform on managed objects (see Figure 1). **Obligation policies** define what a manager *must or must not* do and hence guide the decision making process; the manager has to interpret policies in order to achieve the overall objectives of the organisation.

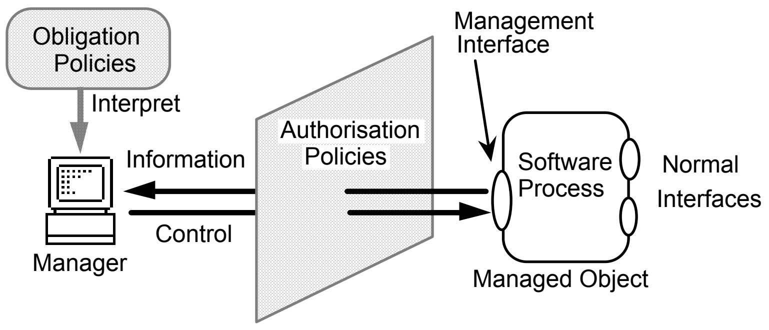

Figure 1 Policies Influence Behaviour.

Human managers are adept at interpreting both formal and informal policy specifications and, if necessary, resolving conflicts when making decisions. However the size and complexity of large distributed systems has resulted in a trend towards automating many aspects of management into distributed components. If the policies are coded into these components they become inflexible and their behaviour can only be altered by recoding. There is thus a need to specify, represent and manipulate policy information independent from management components to enable dynamic change of policies and reuse of these components with different policies.

There may be many different policies relating to the management of a large distributed system, with multiple human managers specifying policy at the same time. The complexity of the problem makes it impossible to prevent conflicts and inconsistencies, so the policy service must support analysis, wherever possible, to detect these and at least warn human users of potential conflicts and inconsistencies.

This paper presents the common policy concepts being used by two Esprit funded Projects – SysMan and IDSM which are implementing distributed management applications based on the use of domain and policy services.

[1] We consider a communications network to be a distributed subsystem providing a communications service, within an overall distributed system which may include various other services such as storage, directory, time etc. Both services and applications running above them have to managed. The concepts described in this paper can be applied to management of networks, telecommunication systems or any other distributed systems.

2 Management Framework

In this section we explain the concept of domains which are important for grouping objects to which policies apply and show how management applications, domain and policy services fit within an overall management architecture.

2.1 Domains

Management of a distributed information system cannot be centralised in a single human or automated entity but must be distributed to reflect the distribution of the system being managed. Management must thus be structured to partition and demarcate responsibility amongst the multiple managers. This structuring could reflect physical network connectivity, structuring of the distributed application or possibly reflect the hierarchical management structure (for example corporate headquarters, regional, site, departmental, and section management) found in many organisations. There will be a variety of managers fulfilling different functions and operating in different contexts, but having responsibilities for the same object. For example the maintenance engineer and the user of a workstation have different management responsibilities for the same workstation. The management structure must be able to model these overlapping responsibilities. Domains provide the framework for partitioning management responsibility by grouping objects in order to specify a management policy or for whatever reason a manager wishes.

A **management domain** is a collection of managed objects which have been explicitly grouped together for the purposes of management. More concretely, a domain is a managed object which maintains a list of references to its member managed objects. If a domain holds a reference to an object, the object is said to be a **direct member** of that domain and the domain is said to be its **parent**.

Since a domain is itself a managed object, it may be a member of another domain and is said to be a **subdomain** of its parent. Subdomains are the means of flexibly partitioning a large group of objects and applying different policies to different subgroups or assigning responsibility for applying policy to different managers. Members of a subdomain are **indirect** members of the parent domain. Managed objects can be direct or indirect members of multiple domains. When an object is a direct member of multiple domains, the parent domains are said to **overlap**. Overlapping domains thus have one or more member objects in common. Domains are similar to the notion of a directory commonly found in hierarchical file systems. More information on domain concepts can be found in [1,2,3] and a detailed specification of the domain service in [4].

2.2 Management Architecture

Figure 2 shows the overall distributed management system architecture. Rather than implement a single monolithic management application (MA) to perform all aspects of management, we have an extensible set of management applications which have a consistent user interface. These make use of a common set of underlying management services for monitoring and manipulating domains and policies. The management objects may interact using various communication services to meet the requirements of particular applications.

Each MA may have its own managed objects grouped into domains and may have one or more human managers depending on the scale of the application and the need for partitioning responsibility. A manager “sees” all the objects (within domains) for which he is responsible or can access. This is analogous to accessing files and devices in a Unix system via the hierarchical Unix directory. A MA will have a user interface (UI) specific to that application but the UI, dealing with browsing the domain hierarchy, and specifying policy will have a common "look and feel" across applications.

Although Figure 2 shows a layered hierarchy, this should not be taken too literally. For example there is both a management application part and a service part for configuration, security and monitoring. The management applications are themselves distributed and may directly access distributed processing or communication services without using intermediate layers. The common management services and the distributed processing services may also be implemented by distributed components. The communication system and the distributed processing services should themselves be managed by the management applications they support. All management applications interpret and apply policies and are subject to security to control access. Further information on the Management Architecture can be found in [5] and how it is being implemented in the IDSM project in [6].

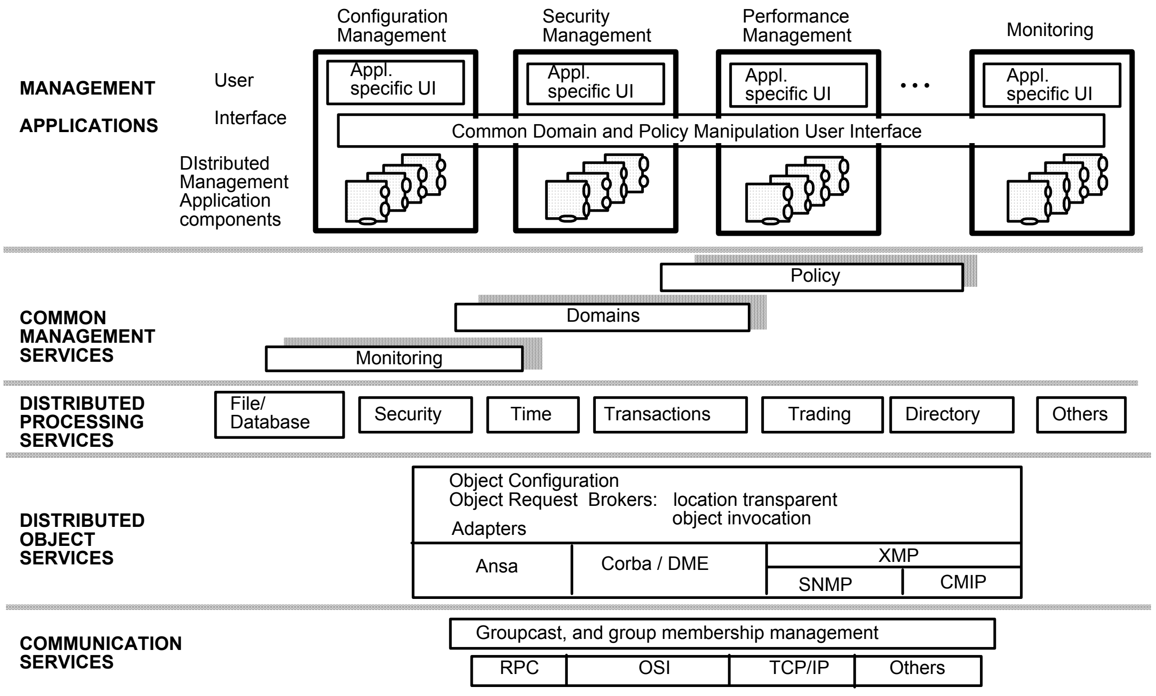

Figure 2 Distributed management system architecture

3 Management Policy

In this section we elaborate on the concepts of policy introduced in section 1 and show how domains can be used to specify the scope of a policy.

3.1 Policy Classification

In an object oriented approach, the external behaviour of an object defines how it interacts with other objects in its environment. We refine the concept of policy to be the information which influences the interactions between a **subject** and a **target** and so the policy specifies a relationship between the subject and target. Multiple policies may apply to any object as it may be the subject or target of many policies.

3.1.1 Authorisation Policy

Authorisation policy defines what activities a subject is permitted to do in terms of the operations it is authorised to perform on a target object. In general an authorisation policy may be positive (permitting) or negative (prohibiting) i.e. not permitted = prohibited. Authorisation policies are considered *target based* in that there is a reference monitor associated with the target which enforces the policy and decides whether an activity is permitted or prohibited. We do not consider mandatory military type policies in this paper.

Activity based authorisation

The simplest policies are expressed purely in terms of subject, target and activity:

* John is permitted to read file F1 (Positive)
* John is prohibited to read, write or execute file F3 (Negative)

A target based reference monitor can then make a decision based on the subject and operation although an implicit subject may be specified.

* Any object is permitted to read file F1 (Positive)
* Any object is prohibited to write file F3 (Negative)

State Based authorisation

State based authorisation policies include a predicate based on object state (i.e. a value of an object attribute) in the policy specification. These are common in database access control and safety critical systems:

* John is permitted to read personnel records where employment grade <10
* The operator is prohibited from performing close_valve on reactor when reactor.Temp > 100
* Managers with current location = planning office are permitted to read expansion plans
  (i.e. they are prohibited when visiting other locations – this assumes current location is an attribute of a manager).

3.1.2 Obligation Policies

Obligation policy defines what activities a subject must (or must not) do. The underlying assumption is that all subjects are well behaved, and attempt to carry out obligation policies with no freedom of choice. This may be true of automated subjects but will in general not be true of human subjects. Obligation policies are *subject based* in that the subject is responsible for interpreting the policy and performing the activity specified.

Activity Based Obligations

Simple obligation policies can also be expressed in terms of subject, target and activity, but may also specify an event which triggers the activity.

* The company director must protect the assets of company XYZ (Positive)
* On error count > 100 monitoring agent must send warning message to operator (Positive, event triggered)

* The standby manager must not perform any control actions (Negative)
* Employees must not talk about their jobs to the press (Negative)

State Based Obligation

An obligation may also be specified in terms of a predicate on object state. In some cases this can be used to select subject or target objects to which an obligation policy applies.

* Controller must control boiler temperature such that 50 < boiler.temp > 100
  (Positive obligation in terms of target state)
* Managers must perform reset on links with error count > 50
  (Positive obligation on selected targets based on state)
* Managers with version < 1.5 must not perform diagnostic test A500
  (Negative obligation applying to selected subjects)

3.1.3 Discussion

Authorisation policies are specified to protect target objects and are usually implemented using security mechanisms in the operating system as subjects cannot be trusted to enforce them. Obligation policies are implemented by the management system i.e. interpreted by managers which must be trusted. Authorisation policies are less dynamic than obligation policies. For example obligation policies may be triggered by an event which results in an action being performed but are effectively dormant until the event occurs again. We have not identified any need for event based authorisation policies.

A negative obligation may appear to be the same as a negative authorisation but the responsibility for preventing the activity lies with the subject rather than with a target based reference monitor. This assumes the subject is well behaved and trusted. The subject may in fact be authorised to perform the activity and the negative obligation is activated to stop it on a temporary basis. For example, a standby manager may normally be authorised to perform control actions but a negative obligation stops the standby manager. Although it would be feasible to transform a negative obligation into a negative authorisation, this may be inconvenient due to the controls and overheads involved in introducing authorisation policies into the system. Transformation to an authorisation policy is necessary if the subject cannot be trusted to perform a negative obligation.

State based policies are more difficult to implement than activity based policies. A reference monitor would have to query subject or target objects to check their state in order to determine whether to permit an action for authorisation policies [7]. A state based obligation policy e.g. to maintain boiler temperature between 50 and 100 cannot be directly interpreted by an automated manager as it needs "intelligence" to work out how to achieve the required goal. The obligation could be refined into the following activity based obligations (which also allow for some time lag in the boiler heater affecting temperature).

* On boiler.temp < 52 controller must switch on boiler heater
* On temperature > 98 controller must switch off boiler heater

Negative state based policies can sometimes be transformed into positive ones by modifying the predicate. For example

* The operator is permitted to close valve on reactor when reactor.Temp ≤ 100

is equivalent to the negative policy defined in 3.1.1 above. This assumes there is an implicit negative authorisation policy forbidding any access unless a positive authorisation policy permits it. Combining positive and negative policies can result in conflicts [8].

Wies [9] has a similar policy mode classification to the above but extends this with additional classification criteria such as lifetime, geographical scope, organisational structure, type of service, type or functionality of targets, management functionality to which the policy applies. These criteria are then used to derive attributes for a policy template.

3.2 Policy Constraints

A constraint can optionally be defined as part of a policy specification to restrict the applicability of the policy. It is defined as a predicate referring to global attributes such as time or action parameters, as explained below.

Temporal Constraints specify time limits before, after or between which a policy applies e.g. between 09.00 and 17.00 or before 31 December 1994. They may be used to specify a validity time or expiry time for a policy.

Parameter value constraints define permitted values for management operations. For example the security policy that passwords must be greater than 6 characters in length and contain at least one non-alphabetic character can be considered a constraint on a change password operation parameter.

Preconditions could define the resources which must be available for a management policy to be accomplished. For example, a dynamic load balancing policy could specify that processes may be migrated to a machine in domain D1 with load < 60% up to limit of 4 processes per machine. Budget allocation is often considered a policy decision.

Other constraints which limit the applicability of a policy can be defined as part of a selection expression for a state based policy based on object attributes to select the set of subject or target objects within a domain to which the policy will be applied (see section 3.4 below).

3.3 Policy as Relationship Objects

Policies encapsulate a representation of information affecting component behaviour so we treat them as objects which provide operations for querying or changing policies [10]. A policy service then provides the operations for creating, deleting, storing and retrieving policy objects. Policy scope is specified using domains, so the policy service must also provide the ability to identify what policies apply to a domain and then use the domain service to identify the objects within the domain. There are advantages in treating policies as managed objects and structuring them into domains, so that an authorisation policy can be defined to control which managers are permitted to modify a set of policies or to define "meta policies" about policies (see section 4.4.).

Another reason for an object oriented approach to policy specification is that it is useful to be able to define a policy class which defines most of the attributes of a particular policy. When specifying policies for a particular application, multiple instances of that policy can then be created, with remaining policy attributes being defined for the particular instance. The policy class is like a template with values for specific attributes being provided when a policy instance is created.

Policies are not active objects in that they do not instigate management operations. Managers are the active objects which are responsible for interpreting an obligation policy and performing the activities specified. A reference monitor uses information such as an access control list derived from authorisation policies to decide whether to permit an operation [11].

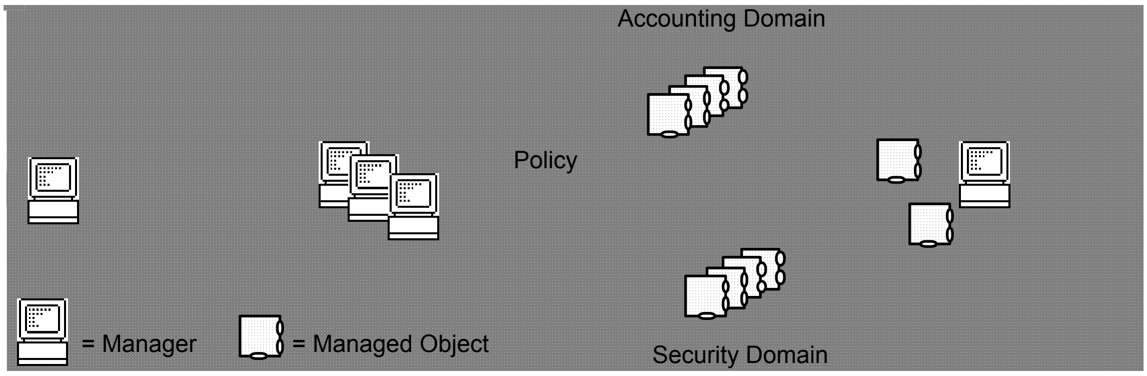

Figure 3 Typical Management Relationships

Figure 3 also shows that a policy may specify a reflexive relationship, whereby managers are members of the managed domain and so could be both subjects and targets of the management policy. This reflects the fact that managers may manage themselves in some circumstances e.g. authorised to approve their own expenses. There is no restriction on the type of object within a single domain so the policy may need to specify the type of object to which it applies unless it is applicable to all objects in the domain.

The OSI Manager, agent, managed object relationships [12] can be modelled by 2 sets of policies – policies which specify the relationship between the Manager and the agent and those which specify the relationship between the agent and the managed objects for which it is responsible. The latter are probably implicit as OSI management does not really consider agents to be intelligent and make management decisions.

3.4 Policy Scope Specification

In very large systems, the number of objects is so large that it is impractical to specify policies for individual objects. Instead it should be specified for sets of objects. The set of objects to which a policy applies could be specified in terms of object attributes e.g. a particular type of object or those objects in a particular state. A search over all reachable objects, within a distributed system, to determine these sets is impractical. The number of reachable objects within a large scale distributed system, is potentially millions and is not known a priori. The selection of objects is thus limited to be within the **scope** of a domain. This limits the search space for object selection to a predefined set to make sure the selection terminates within a defined time. For example the policy

* Kevin must install new kernel on workstations with workstation type=Sparc IPX

is impractical as there are potentially millions of such workstations connected to the internet, so the scope should be limited as in the following policy.

* Kevin must install new kernel on workstations in domain dse.doc.ic.ac.uk with workstation type=Sparc IPX

Another advantage of specifying policy scope in terms of domains is that objects can be added and removed from domains to which policies apply without having to change the policies.

3.4.1 Propagation to Subdomains

Policies apply to sets of objects within domains, but domains may contain subdomains. To avoid having to respecify policy for each subdomain, policy applying to a parent domain, should **propagate** to member subdomains of the parent. For example a policy applying to an organisation should also apply to departments within that organisation. A subdomain is said to **inherit**, the policy applying to parent domains (but this is not the same as object oriented inheritance). With policy propagation, a policy specified for a domain is applied recursively to all direct and indirect members of that domain. For example, in Figure 4 the policy specified between D1 and D2 will propagate to managers in D2 and the managed objects in D4 and D5.

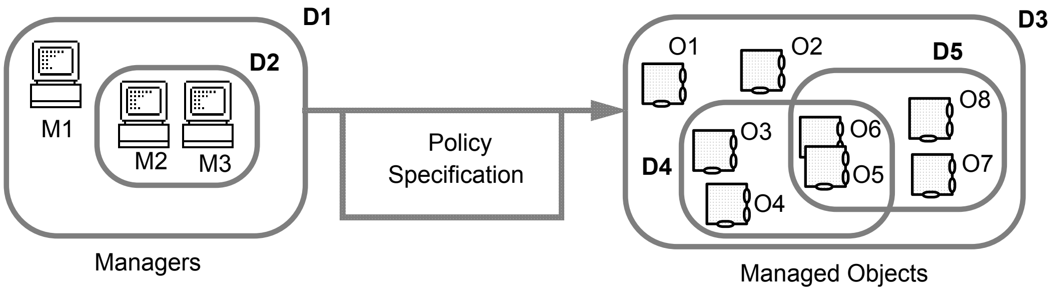

Figure 4 Policy Propagation

It should be possible to override the default policy propagation either at the policy or domain level. A policy may specify that it applies only to direct members of the target or subject domains. A domain attribute may specify that any policy applying to the domain does not propagate to indirect members, irrespective of what is specified in the policy.

In order to efficiently determine the policies applying to a domain and hence to an object within it, the domain must hold references to those policies.

3.4.2 Set Selection

A policy should be able to select the set of subject or target objects within a domain to which it applies using a predicate based on the values of object attributes. The simplest case occurs when a policy applies to all objects in a domain. The set of objects to which a policy applies has to be evaluated at the time the policy is interpreted because domain membership can change dynamically. Object selection is based on *Scope Expressions* which define a (possibly empty) set of objects and are based on combinations of the following:

i) The object itself.

ii) Direct and indirect members of the domain i.e. policy is applied recursively to all subdomains which are members or indirect members of the domain.

iii) Limited propagation. i.e. policy is applied recursively to a limited number of levels of subdomains which are members the domain.

iv) A predicate based on object attributes is used to select objects. For example the policy applies to objects of a particular type or in a particular state. A *location constraint* may limit the applicability of an authorisation policy in terms of the location from which operations on the objects can be invoked e.g. a file can only be read from terminals in a particular office. Evaluating a set of objects, using a predicate based on an object attribute value together with policy propagation, could be very expensive to implement in a distributed system.

v) A set expression in terms of members of the domain can be evaluated to give a set.

vi) Any objects - this allows an implicit scope. For example any manager may be authorised to perform an operation on an object. "Any" is only permitted as the subject scope for an authorisation policy or the target scope for an obligation policy.

The use of "any" as the subject of an obligation policy does not make sense as no specific subject has the responsibility to carry out the actions. We have not identified a use for "any" as the target for authorisation policy.

3.4.3 Scope Expressions

Scope Expressions always return a set of objects and are defined as follows:-

```
SE ::=       "ANY" | SC_EXPR

SC_EXPR ::= object |
            { object } |
            *object |
            * NUMBER object |
            SC_EXPR + SC_EXPR |
            SC_EXPR - SC_EXPR |
            SC_EXPR ^ SC_EXPR |
            select( pred, SC_EXPR ) |
            ( SC_EXPR )
```

Operators

+	set union
-	set difference
^	set intersection
*	this returns a set that contains all direct and indirect members of the domain if it is applied on a domain object ; otherwise it returns a set that contains the object itself.
* NUMBER	a set that contains all direct and indirect members of the domain as far down as the NUMBER'th level if it is applied on a domain object returns. That is, *1D1 gives the set of direct members of domain D1. If applied to an object it returns a set that contains the object itself.
{ }	returns a set that contains the object on which it is applied i.e. it converts a single object to a set containing that object. It is only needed for set theory consistency, as the operators (+, -, ^) are only applied to sets of objects. There is no ambiguity if it is omitted.
object	shorthand version or saying { object }
select(*pred***,** *sc_expr* **)**	returns a sub-set of the set returned by *sc_expr* with the members of the selected set determined by the predicate. The predicate will typically be a function which is applied to all members of the set returned by *sc_expr*.

The interpretation of the expressions is from left to right.

Operators can be divided into two categories. The first category includes the set operators (+, -, ^) which are applied to sets of objects. The second one includes the object operators (*, { }) that are applied to objects and return a set of objects. The set that the object operators return is evaluated by traversing the domain hierarchy starting with the domains referenced in the expression.

For example, referring to Figure 4:

- `*1D4-O3+*1D5` = union of direct members of D4 (except O3) and D5 = `{O4,O5,O6,O7,O8}`
- `*D4^*D5` = intersection of direct and indirect members of D4 and of direct and indirect members of D5 = `{O6, O5}`.
- `*D3` = all direct and indirect members of D3 = `{O1,O2,O3,O4,O5,O6,O7, O8,D4,D5}`
- `select(type!=Domain, *1D4-O3+*1D5)` = non-domain members of the set of the union of direct members of D4 (except O3) and D5 = `{O4,O5,O6,O7,O8}`.

4 Example Policy Objects

A policy object specification defines the following attributes:

i) Modality: positive or negative authorisation, positive or negative obligation (i.e. A+, A-, O+, O-)

ii) A subject which defines one or more manager domain scopes

iii) A target which defines one or more managed domain scopes affected by the activity

iv) An activity which define a set of actions or permitted operations

v) Constraints which apply to the activity

Example policies with these characteristics are given below.

4.1 Access Rules

An access rule is a simple example of a management authorisation policy which specifies a relationship between managers (in a subject scope domain) and managed objects (in a target scope domain) in terms of the management operations permitted on objects of a specific type. The access rule may also define constraints on these operations (see 3.2 above) and make use of scope expressions to select subsets of the objects within the subject or target domains. In Figure 5, operations OpA and OpB are permitted on objects of type T1 and operations OpX and OpZ on objects of type T2. These operations can only be performed between hours of 09.00 to 17.00. Examples of the use of access rules for service specification can be found in [2].

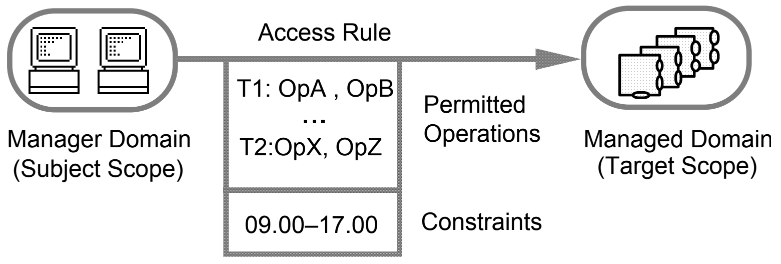

Figure 5 Access Rules

4.2 Domain Membership Policy

A manager can specify the initial membership of a domain by specifying an object selection predicate for searching a database or another domain, but this is not provided as part of the basic domain service. Membership policies are then needed to constrains the objects which can be subsequently created in the domain or included from another domain. Other membership policies relate to the number of objects permitted in a domain. Example membership policies are given below.

- Only objects which implement a particular interface type can be members of the domain i.e. any subject is permitted to include or create objects of type T in target domain Dt.

  ```
  A+ any {include X, create X} Dt when X.type=T
  ```
- A domain of managers can have only a single manager to prevent conflicts between multiple managers.

  ```
  A+ any {include, create} Dt when Dt.membernum = 0
  ```
- There must always be at least two objects in a domain for backup purposes. This also requires an obligation policy for a manager in domain Ds, on receiving a failure event, to delete the failed object and create a new one.

  ```
  A- any {remove, delete} Dt when Dt.membernum > 2;
  O+ on fail(X) Ds {delete X; create X} Dt
  ```
- An object should always be a member of at least one domain if it is to be managed using the domain based management applications mentioned in section 2, so a policy could specify that removing it from a target domain Dt is only permitted if has more than one parent.

  ```
  A+ any {remove X} Dt when X.parentnum > 1
  ```

4.3 Delegation

In some applications a manager may delegate an activity to a proxy manager (or agent) to perform on his behalf. There is a need to control to whom the managers can delegate and what operations they can delegate. This type of policy requires two subject domains – for the delegator and delegatee. The policy shown in Figure 6 permits Managers in Domain D1 to delegate the right to perform operations OpA and OpC on target objects of Type T1 in Domain D2, to a proxy manager in Domain D3. The policy expires after 31 December 1994.

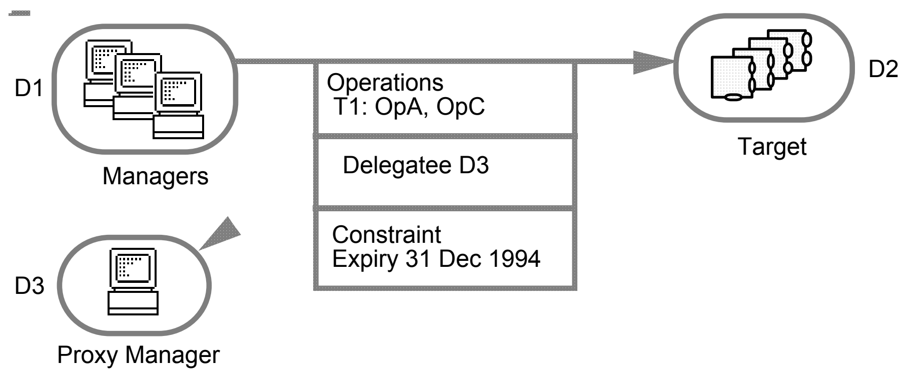

Figure 6 Delegation of rights.

The implementation of this type of policy requires extended access control lists which contain information on delegatees as well as subjects.

4.4 Security Administrator

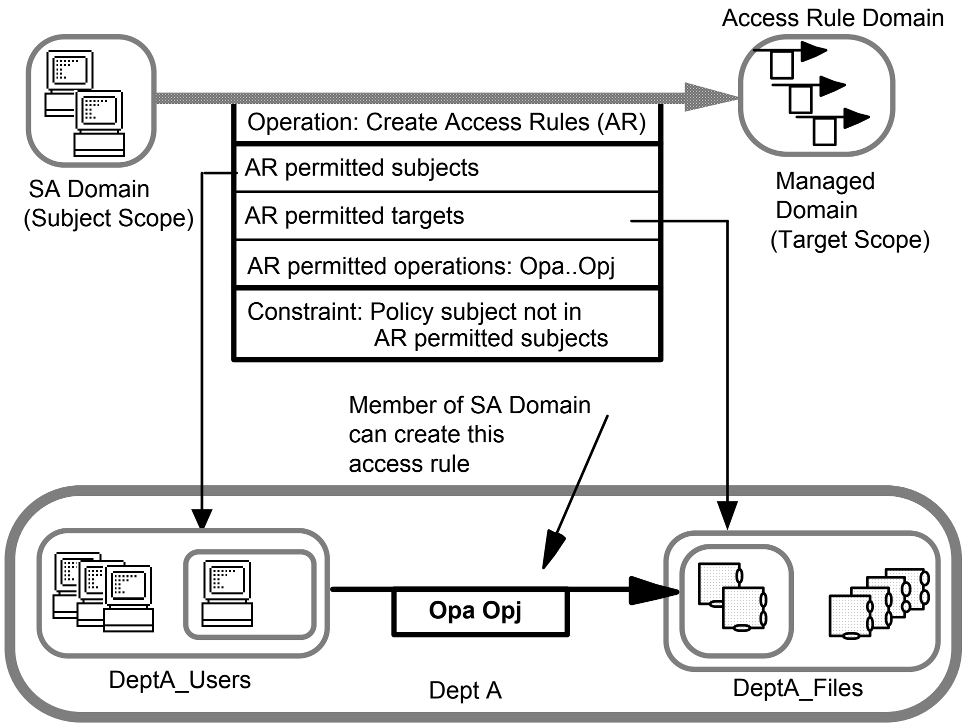

Figure 7 Authorisation Policy for a Security Administrator (SA)

A security administrator (SA) in a commercial environment would typically create access rules (ARs) for other subjects (excluding himself) to access specific target resources. The authorisation policy applying to the SA should limit the subjects for whom he can create ARs, the target objects to which the rules can apply and the operations specified by the ARs. This is a meta-policy about managing policy objects (access rules) and also specifies a relationship between multiple domains. It is the most complicated of the examples. Figure 7 shows there are a number of scopes which limit the ARs which can be created. *AR permitted subjects* specifies to whom access can be given, the *AR permitted targets* define the set of target objects to which access can be given, and an *AR permitted operations* define the set of operations which can be included in access rules. The principle of separation of responsibility means the SA should not be permitted to give access to himself, so the subject should not be a member of the *AR permitted subjects*. If we consider ARs as objects which are created in domains, there is a target domain in which the SA is permitted to create ARs.

Interpreting policy objects such as those relating to a security administrator is considerably more complicated than simple access rules. It would be the function of the policy service to interpret and enforce this policy as it relates to creating policy objects.

4.5 Responsibility

The concept of responsibility can be modelled as an obligation policy. For example consider that manager X has responsibility to update software in a domain of workstations. Manager Y is his superior and has ultimate responsibility to determine the work is carried out correctly. This is modelled using two different obligation policies, one to perform the updating and one to indicate the responsibility relationship as shown in Figure 8.

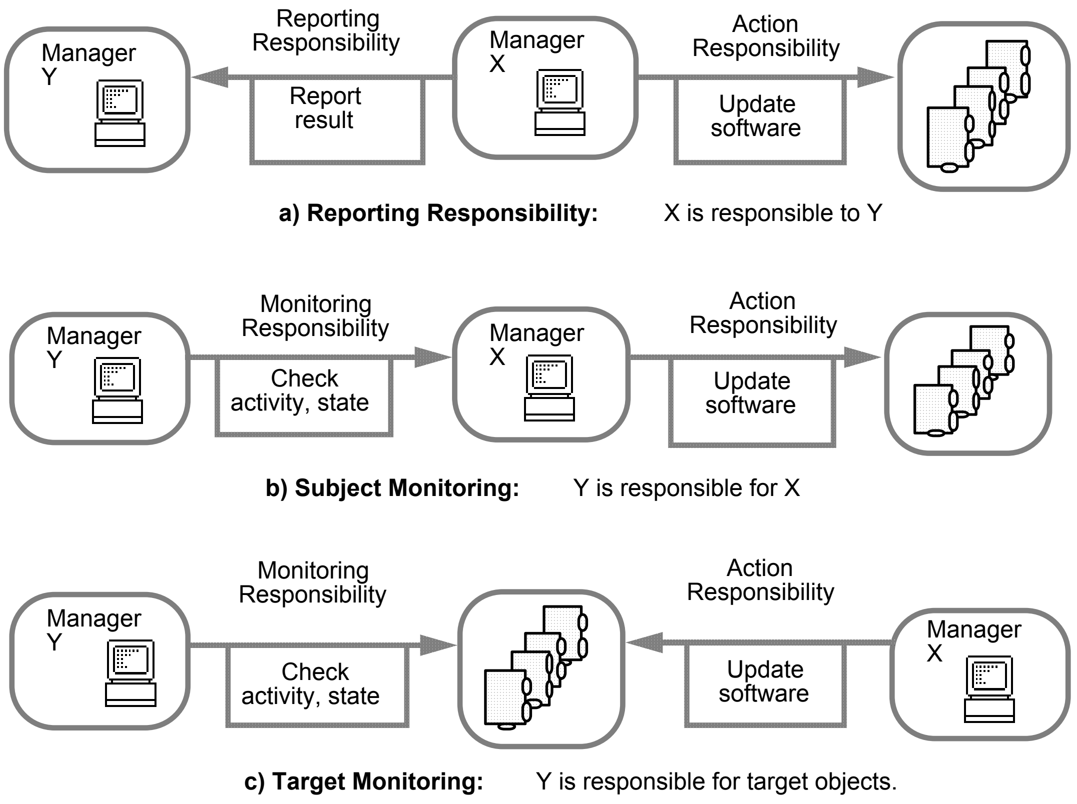

Figure 8 Modelling Responsibility as Obligation Policies.

5 Policy Implementation Issues

A **Policy Dissemination Function** transforms policies into a form suitable for interpretation or enforcement and sends obligation policies to managers in the subject domain and authorisation policies to reference monitors associated with objects in the target domain. An example transformation is an authorisation policy object into an access control list (ACL) entry or capability. ACLs are stored with the target domain and have to be propagated to nested subdomains [13]. The authorisation policies have been applied to specifying service access rights for cellular networks [2].

The generalised policy concepts were derived from our initial work on access control policy, so the authorisation policy aspects are further advanced than the obligation policy concepts. We are experimenting with a notation which can be used for both authorisation and obligation policies as they have similar attributes but the respective implementation mechanisms are very different. Obligation policies are of the form

O+ | O- [on <event>] <subject> {actions} <target> [when <constraint>]

where the action is triggered by an event occurring and the action could be specified by a C language procedure. The constraint is a predicate which is evaluated when the event occurs and can be used to inhibit the action being performed. The event and constraint expression are optional. In particular this has been applied to a monitoring service where these policy rules can be used to combine and filter events or generate higher level event reports [14]. We are developing graphical tools for specifying both authorisation and management policy.

The IDSM partners are using their proprietary Network Management platforms to implement policies and domains as OSI managed objects in special Management Information Bases (MIBs) [6, 15] supported by service managed objects. These implementations use the Network Management Option within OSF's DME to access OSI and SNMP managed objects via the XMP interface [16]. A management agent uses the authorisation policies to make access control decisions for management operations. Reporting obligation policies for generating event messages are being translated into event forwarding discriminator managed objects. The SysMan project is using ANSAware [17], which is a distributed object oriented programming environment so domains, policies and the distributed servers which store them can be directly implemented as Ansa objects. One of the commercial partners is porting this implementation to a CORBA platform [18].

6 Manager Roles

6.1 Conceptual Issues

The concept of a role is well understood in enterprise modelling and there is an extensive literature relating to role theory [19]. Role Theory postulates that individuals occupy positions in an organisation. Associated with each position is a set of activities including required interactions that constitute the role of that position.

A **manager role** is defined as the set of authorisation and obligation policies for which a particular manager position is the subject. A role thus identifies the authority, responsibility, functions and interactions, associated with a position within an organisation. Example manager positions could include managing director, security administrator, operations manager, operator responsible for North Region. A person may be assigned to one or more roles and multiple individuals can share a single role.

A **manager position** defines a particular position within an organisation, such as financial manager, managing director, to which different people may be assigned over a period of time. The role refers to a manager position rather than a particular person, because people are frequently assigned to new roles, and it would be very time consuming to change policies which reference that person. Within the management environment the functions and authority of a human or automated manager is defined in terms of the obligation and authorisation policies which apply to the manager position. This defines the overall functionality of the position. A role may also relate to an automated manager, although it is less likely to be frequently reassigned to new roles.

Policies which have propagated down to a position, but do not explicitly reference the position are not included in the set of role policies. Propagated policies are part of the organisation policies as they may also apply to different roles or objects, and are not specific to the manager position which is the subject of the role.

It would be very useful to be able to parameterise a role with specific positions and target domains. It would thus be possible to define the role policy set as a class from which particular instances can be created. For example a role could be defined for a region manager and this could be used to create North, South, East and West region manager roles. Each of the 4 roles instances relates to different manager positions each with their own specific target domains, but specifies the same policy activities and constraints for each manager position.

6.2 Implementation Issues

It is assumed that the humans occupying roles will perform management functions related to the distributed system and so have to be represented within the system by an adapter object which interacts with a suitable presentation device (workstation or terminal). We now show how the domains described in section 2.1 can be used to represent Users and Positions.

The policies applying to a person are defined in terms of a **User Representation Domain (URD)** which is a persistent representation of the person or human manager. When the person logs into the system an adapter object is created within the URD to interact with the person's workstation. The policies specified for the domain apply to the adapter object representing the person (c.f. DME adapters [16]).

Policies relating to a role should be independent of the person occupying that role so should not reference a URD. Instead, a **Position Domain** is created and policies pertaining to the role are specified with the position domain as a subject (see figure 9). Allocating a person to a position is accomplished by including his URD within the position domain. Role policies propagate to the URD and hence apply to the manager adapter object. The manager may be a member of multiple position domains if performing multiple management roles. Multiple URDs may be included in a position domain indicating a shared position but obviously this would require the managers to coordinate their activities via a suitable protocol.

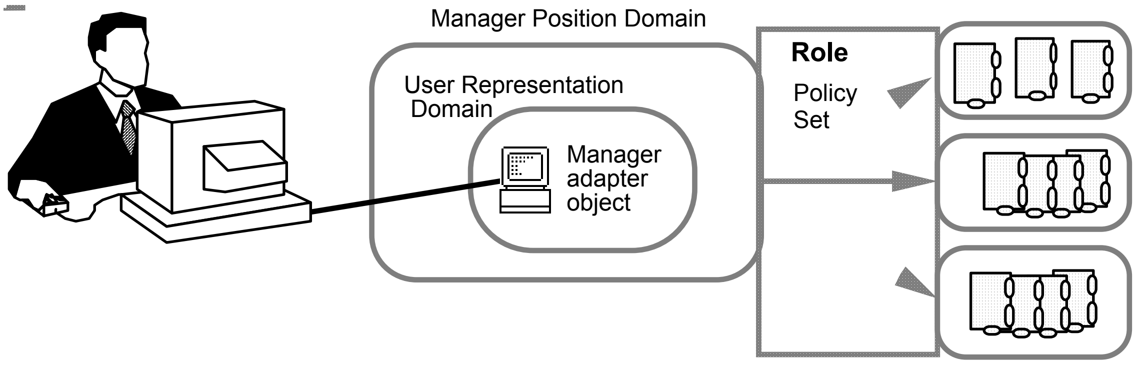

Figure 9 Position Domains and Roles

The set of policies associated with a role could be stored in a single domain for ease of reference, by the manager, but this is not shown in figure 9. However propagated polices would not be defined specifically for the position domain so may be stored elsewhere. The issues related to storing policies in domains requires further study.

The implication of parameterised roles is that it must be possible to define a policy class object which can have some attributes predefined but others can be defined as parameters which are provided when a policy instance is created.

7 Policy Hierarchy

Policy can be expressed as a hierarchy where a high level policy goal can be refined into multiple levels of lower level policy and eventually into a set of policy rules [20].

- **Policy goals** define activities in terms of actions and operations which have to be transformed or refined before they can be executed by managers or target managed objects. Policy goals will have to be interpreted by human managers or, in some cases, by an expert system capable of application dependent goal refinement.
- **Policy Rules** define activities in terms of actions or operations which can be directly executed by manager or managed objects c.f. access rules. These could be interpreted by automated tools.
- **Policy Mechanism Information** may be generated from policy rules for efficient implementation of policy mechanisms e.g. transforming access rules into access control lists or capabilities for actually controlling access at run time [11,13, 15].

Both goals and rules can be expressed as obligation or authorisation policies and be represented by a policy object with the attributes of subject, target, constraints and activities. A high level obligation or authorisation policy may be refined into a number of lower level obligation and authorisation policies. For example a high level goal:

```
* O+ Manager M must protect Dept. D files from loss due to
  fire or media failure
```

could be refined, via many intermediate more detailed policies, into the following set of implementable ones in which an archiver (in Domain BackupSW) performs a backup to tape followed by one to a remote store. Authorisation policies are needed to permit backup software to read and mark files and to perform file transfer to the remote store address passed as a parameter.

```
* O+ At 22.00 every Thursday archiver {tape_backup} Domain
  Dfiles
* O+  At 03.00 every Friday archiver
  {remote_backup (safestore1.ic.ac.uk)} Domain Dfiles
* A+ Domain BackupS/W{read, write} Domain Dfiles
* A+ Domain BackupS/W {file_transfer, file_access} Domain
  safestores
```

The distinction between goals, rules and mechanisms is not fundamental. What is considered a goal at one time may eventually become a rule or a mechanism with improvements in technology and implementations techniques. Thus we use the same concept of a policy object for all levels of abstraction in the policy hierarchy, but some policies may only be interpretable by humans.

The motivation for understanding hierarchical relationships between policies is to determine what is required for the satisfaction of policies. If a high-level policy is defined or changed, it should be possible to decide what lower-level policies must be created or changed. The ultimate aim is to be able to specify high-level policies and automatically generate the lower-level ones but this is similar to automatically generating code from abstract requirements specifications and this is very difficult to achieve.

Another motivation is for analysis to see whether the set of lower-level policies actually fulfil the higher-level policy, by providing complete cover over all the target objects and actually meeting the policy goals. Again this can be very difficult to achieve. The issues relating to refinement and analysis of Policy Hierarchies are discussed in [21]. Other discussions on policy hierarchies can be found in [9, 22].

8 Policy Analysis

In general multiple policies apply to an object. For example an obligation policy will specify an activity that a manager must do and there should be a corresponding authorisation policy permitting the manager to perform the activity. However a set of policies may cause problems, because of omissions or incorrect policy specifications. A conflict between policies may result in a manager being unable to perform its activities. The managers may not attempt to perform the activities they are permitted to do or they may be sufficiently intelligent to avoid conflicts, as is often the case with human managers. Managers need tools to analyse the set of policies stored in the policy service to detect any of the following conditions which may otherwise result in problems within the management system.

i) Coverage

This involves checking whether derived lower level policies cover all the objects specified in a higher level policy. The simplest case occurs when a target set is partitioned by creating multiple policies, corresponding to the same activity as the original but applying to a subdomain of the original target domain. It is comparatively easy to check that every member of the original policy's target set is a member of at least one target set of derived higher level policies. Determining that at least one authorisation policy applies to every object so that it can be managed should also be practical. Coverage analysis is a subset of the more general problem of completeness analysis which is notoriously difficult to achieve.

ii) Missing Obligation/Authorisation

As obligation and authorisation policies are considered independent, in theory there should be both an obligation and authorisation policy for a manager to perform an activity. The existence of an obligation without a corresponding authorisation for the activity prevents the obligation being performed. However it may be valid for authorisation policies to exist without the corresponding obligation. For example a back up manager may have authorisation policies permitting various actions, but the corresponding obligation policies are created only if the primary manager fails.

iii) Conflicts

A conflict may occur between any two policies if one policy prevents the activities of another policy from being performed or if the policies interfere in some way that may result in the managed objects being put into unwanted states. As the activity of a policy can specify a set of actions, there may also be conflicts between these actions within a single policy. Our model of management policies represents both subjects and target objects as sets of objects. A conflict may occur between two or more policies if there is an overlap between the subject and/or target object sets. The overlap relationship between sets of objects exists when their intersection is non-empty.

Obvious conflicts occur if there is both a positive and negative authorisation or obligation policy with the same subjects, targets and actions. It is not practical to forbid overlaps as they are very useful for many situations. A set of managers may perform different management activities on disjoint target domains, resulting in subject overlap. Target overlaps occur when managers who do not have full responsibility for a particular target domain, are given limited rights to perform some activities (e.g. query state or perform diagnostics) as they make use of a service provided by the objects in the target domain. In many cases, the fact that a conflict *may* occur does not mean it *will* occur. Policy conflicts are discussed in more detail in [8].

The problems of detecting conflicts is extremely difficult. Most existing work relates to authorisation policy [23]. Analysis of the policy objects without any knowledge of the application or activities may detect positive-negative conflicts of modalities and conflicts between obligation and authorisation policies, and so it may be possible to automate this. Most other conflicts require application dependent knowledge about the activities specified in the policies to detect whether there is potential conflict. This is likely to require human intervention. We are experimenting with tools to detect positive/negative conflicts and missing authorisation policy.

9 Related Work

The ESPRIT funded DOMAINS project also worked on Domain based policy management [20, 24, 25]. Their policies and managers were included in the domain being managed. There was a single manger in a domain which had unlimited access to all managed objects in that domain, so they did not really consider authorisation policy. Although some members of the DOMAINS project are working in the IDSM project, the DOMAINS' approach was not considered sufficiently flexible to cope with inter-organisation systems where managers and the objects they may manage are inherently in different domains. The DOMAINS project had the concept of high level goals being translated into policies which were then translated into sets of plans. In our approach, goals are abstract policies interpreted by humans and rules are more concrete ones, possibly interpreted by automated managers. We consider that all types of policies should be represented as objects with subject, target, actions and constraints as attributes and that there may be more than 3 levels of abstraction.

Roos et. al. [26] take a similar approach to our concepts of policies and domains, but their policy objects have two parts. A passive relationship object is very similar to our policy objects and defines a relationship between a manager and target domains. The second part is an active policy object which is a form of proxy manager which tries to achieve the goals specified in the passive policy on behalf of the manager. This active policy object would poll the managed objects in the domain and perform management operations on them. We would model this as a hierarchical management structure with one set of policies defining the manager to proxy relationship and another set of policies defining the role of the manager proxy with respect to the target domains. Wies also has active policy objects for enforcement and monitoring of policy [27]. All these policy specifications, which define a manager or proxy's behaviour, look like management algorithms i.e. the policy is now encoded into proxy managers. We do not think that the policy service should be used as the means of *distributing* the management function. Meyer [28] has a notation for specifying manager behaviour as policies, but it look like a manager programming language. Policies need to be sufficiently abstract to be interpreted by managers or reference monitors so that they can be changed dynamically.

The International Standards Organisation (ISO) WG4 are trying to define standards for domains and policies [29, 30], but the work is still very unstable. Our work has been provided as input to this committee, but it is our view that as there is no consensus on what constitutes management policy, it is inappropriate to be trying to standardise concepts for which research is in its infancy.

There are a number of groups working on aspects of security policy [31,32] but they do not cater for large scale distributed systems. The Miro tools [23] provide a graphical means of specifying an access matrix for file system security. This notation is very similar to the diagrams shown in this paper. They also permit positive and negative authorisation policies and they have a tool which checks for ambiguities or conflicts. There is a constraint language which limits the set of diagrams to be those which are realisable or acceptable. This can be used to define a "meta" policy about what authorisation policies are acceptable for a particular site. The Miro tools appear to be the most developed but our experience from experimenting with them is that they are very slow and could not be used for a large number of policies. They have not been used for obligation policies but they are quite general and probably could be adapted for this use. They do not have any support for distribution.

Jonscher's work on modelling access behaviour of database users also has some similarity to our approach [33]. His access rights compare to our authorisation policies and his normative rights have some similarity to our obligation policy although he models both duties and (liberties) freedoms to do actions. We are not convinced of the need for liberties in management policy. We intend to take a similar approach, for our obligation policies, to the triggered actions he uses for duties and liberties.

Lomita is a rule based language for programming the management layer in the Meta system [34]. Lomita rules are of the form **on** *condition* **do** *action,* which is also similar to Jonschers triggered actions, but there is no explicit subject or target – they are defined implicitly.

The Methods specification language (MSL) [22, 35] was developed for specifying policies such as those relating to scheduling for large mainframes, but has not been applied to distributed systems. It takes an artificial intelligence approach of choosing from a set of potentially conflicting goals by assigning priorities to the goals. The implementation makes use of an object oriented database to hold the information for policy based decision making.

There are various forms of Deontic Logic which have operators that denote obligation and permission, which at first sight seem very similar to our obligation and authorisation policy [36]. In most of these obligation implies permission or sometimes permission for an action *means* not being obliged to refrain from an action. In our approach obligation and authorisation policies can be specified independently, although obligation without authorisation can lead to conflicts.

A more detailed survey of policy specification can be found in [37].

10 Conclusions

Policy driven management provides the basis for dealing with automated management of large scale distributed systems and networks. The specification and manipulation of policy will become one of the key research area in the next few years.

This paper has shown that the concept of a passive policy object can be used to model a wide range of authorisation and obligation policies. This policy object defines a relationship between one or more subject and one or more target domains; specifies the actions the subject performs on the target and specifies a constraint to limit the applicability of the policy. The same concept can be applied to high-level (unimplementable) goals interpreted by humans and to low level implementable rules to be interpreted by automated objects.

A policy specification language should not be a programming language for implementing proxy managers but should produce a set of rules which can be interpreted by managers for obligation policies and reference monitors for authorisation policies. This is essential to permit easy change of policy, without reimplementation of management components, to change system behaviour.

Domains are used to specify the scope for applying the policy. We anticipate that domains will be used to group objects to which particular policies apply, although a policy can select a subset of members of a domain to which it will apply using domain set expressions or predicates based on object attributes. This simple domain concept can also be used to represent a user or a position within an organisation.

These concepts can be implemented in any object based environment and implementations on OSI based Network management, ANSAware and CORBA platforrms are being produced in the SYSMAN and IDSM projects. The industrial partners have a commitment to integrate these into their relevant commercial products soon after the project ends.

This document has clarified the concepts relating to using policy to influence management of distributed systems. Many issues remain to be solved. Automatically, deriving low level policies from high level ones is as difficult as deriving programs from requirements specifications. Policy analysis, conflict detection and resolution are also important area requiring considerable research.

Acknowledgements

I acknowledge the major contribution of Jonathan Moffett (now at York University) to the development of the ideas expressed in this paper. My colleagues in the SYSMAN and IDSM projects (from Imperial College, AEG, ICL, Stockholm University, EPFL, Bull, Siemens, FhG-IITB, PTT Netherlands, Telesystemes and NTUA) criticised and suggested revisions to some of the initial ideas based on previous work. I gratefully acknowledge financial support from DTI ESF Project (IED/ 4/410/36/002), Esprit SYSMAN (7026) and IDSM (6311) projects.

References

[1] MS. Sloman and J.D. Moffett, Domain Management for Distributed Systems, *Integrated Network Management I,* B. Meandzija and J. Westcott eds., North Holland, 1989, pp. 505-516.

[2] M.S. Sloman, B.J. Varley, J.D. Moffett, K.P. Twidle, Domain Management and Accounting in an International Cellular Network, *Integrated Network Management III (C-12),* H.-G Hegering, & Y. Yemini eds., North-Holland, 1993, pp. 193–206.

[3] J.D. Moffett and M.S. Sloman, User and Mechanism Views of Distributed System Management, *IEE/IOP/BCS Distributed Systems Engineering*, Vol. 1, No. 1, Aug. 1993, pp. 37-47.

[4] K. Becker, U. Raabe, M. Sloman and K. Twidle (eds.), Domain and Policy Service Specification. *IDSM Deliverable D6, SysMan Deliverable MA2V2,* Oct. 1993. Available by FTP from dse.doc.ic.ac.uk.

[5] M. Sloman, J. Magee , K. Twidle, and J. Kramer, An Architecture for Managing Distributed Systems, *Proc. 4th IEEE Workshop on Future Trends of Distributed Computing Systems,* Lisbon, Sep. 1993, pp 40-46.

[6] B. Alpers and H. Plansky, Domain and Policy Based Management: Concepts and Implementation Architecture, *IEEE/IFIP Workshop on Distributed Systems Operations and Management,* Toulouse, Oct. 1994.

[7] J.D. Moffett and M.S. Sloman, Content-Dependent Access Control, ACM *SIGOPS Operating Systems Review*, Vol. 25, No. 2, April 1991, pp. 63-70.

[8] J.D. Moffett and M.S. Sloman, Policy Conflict Analysis in Distributed Systems Management t, Ablex Publishing Journal *of Organizational Computing*, Vol. 4, No. 1, 1994, pp 1-22.

[9] R. Wies, Policy Definition and Classification: Aspects, Criteria and Examples, *IEEE/IFIP Workshop on Distributed Systems Operations and Management,* Toulouse, Oct. 1994.

[10] J.D. Moffett and M.S. Sloman, The Representation of Policies as System Object, *Proc. Conf. on Organisational Computer Systems (COCS 91)*, Atlanta USA, Nov. 1991, SIGOIS Bulletin, Vol. 12 nos. 2&3, pp. 171-184.

[11] J.D. Moffett, M.S. Sloman, and K.P. Twidle , Specifying Discretionary Access Control Policy for Distributed Systems, *Computer Communications*, Vol. 13, No. 9, Nov. 1990, pp. 571-580.

[12] Information Technology – Open Systems Interconnection – Systems Management Overview, ISO/IEC 10040, Nov., 1992.

[13] K.P. Twidle, Domain Services for Distributed Systems Management, PhD Thesis, May 1993, Department of Computing, Imperial College.

[14] M. Mansouri-Samani and M. Sloman GEM: A Language for Generalised Event Management *Imperial College Department of Computing, Research Report DoC 93/49,* Nov. 1993, Available by FTP from dse.doc.ic.ac.uk.

[15] H. Schwingel-Horner and G. Bonn, IDSM Authorisation Policy Specification and Enforcement in a Hierarchical Management Environment, *IEEE/IFIP Workshop on Distributed Systems Operations and Management,* Toulouse, Oct. 1994.

[16] The OSF Distributed Management Environment architecture. Open Software Foundation, 11 Cambridge Center, Cambridge, MA 02142, USA, May 1992.

[17] ANSAware 4.1: Application Programming in ANSAware, Document RM.102.02, Architecture Projects Management , Poseidon House, Castle Park, Cambridge CM3 0RD, UK, Feb. 1993.

[18] Object Management Group, The Common Object Request Broker Architecture (CORBA) and Specification V1.1, OMG, Dec 1991.

[19] E. Thomas and B. Biddle, Role Theory: Concepts and Research, *Krieger Publishing,* 1979.

[20] Esprit Project 5165 - DOMAINS Basic Concepts, version 2.0 (Nov 1991), *Philips Gmbh,* PO Box 1980, W 5100 Aachen, Germany.

[21] J.D. Moffett and M.S. Sloman, Policy Hierarchies for Distributed System, *Proc. IEEE JSAC*, Vol.11, No. 9, Dec. 1993, pp. 1404-1414.

[22] M.J. Masullo and S.B. Calo, Policy Management: An Architecture and Approach, *Proc. IEEE Workshop on Systems Management,* UCLA, California, April 1993.

[23] A. Heydon, M. Maimone, J. Tygar, J. Wing, and A. Zaremski, Miró: Visual Specification of Security, *IEEE Trans. on Software Eng.*, Vol. 16, No. 10, pp. 1185–1197, Oct. 1990.

[24] Esprit Project 5165 - DOMAINS Deliverable 2c version 1.0, DOMAINS - Management Architecture, *Philips Gmbh,* PO Box 1980, W 5100 Aachen, Germany, May 1992

[25] K. Becker and D. Holden, Specifying the Dynamic Behaviour of Management Systems, *Journal of Network and Systems Management*, Vol. 1, pp. 281-298, Sep. 1993, Plenum Press

[26] J. Roos, P. Putter, and C. Bekker, Modelling Management Policy Using Enriched Managed Objects, *Integrated Network Management III (C-12)*, H.-G Hegering, & Y. Yemini eds., North-Holland, 1993, pp. 207–215.

[27] R. Wies, Policies in Network and Systems Management — Formal Definition and Architecture, *Journal of Network and Systems Management*, Vol. 2, No.1 pp.63-83, March 1994, Plenum Press.

[28] B. Meyer, C. Popien, Defining Policies for Performance Management in Open Distributed Systems, *IEEE/IFIP Workshop on Distributed Systems Operations and Management,* Toulouse, Oct. 1994.

[29] Information Technology – Open Systems Interconnection – Systems Management Overview – Amendment 2: Management Domains Architecture, PDAM 10042, Nov. 1993.

[30] Information Technology – Open Systems Interconnection – Systems Management – Part 19: Management Domain and Management Policy Management Function, ISO/IEC CD 10164-19, Jan. 1994.

[31] D. Brewer and M. Nash, The Chinese Wall Security Policy, *Proc. IEEE Symposium on Security and Privacy,* IEEE Computer Society, 1989.

[32] D. Clark, and D.R. Wilson. A Comparison of Commercial and Military Computer Security Policies, *Proc. IEEE Symposium on Security and Privacy,* 1987.

[33] D. Jonscher, Extending Access Control with Duties Realised by Active Mechanism, *IFIP WG 11.3 Sixth Working Conference on Database Security,* Vancouver, Aug. 1992.

[34] K. Marzullo, R. Cooper, M. Wood, K. Birman, Tools for Distributed Application Management, *IEEE Computer,* Vol. 24, No. 8, Aug. 1991, pp. 42-51

[35] M.J. Masullo and E. Mozes, A Methods Specification Language for object oriented Databases, *Research Report 16360,* 1990, IBM TJ Watson Research Center, Yorktown Heights, New York.

[36] K. Ong, R. Lee, A Logic Model for Maintaining Consistency of Bureaucratic Policies, Proc. 26th Annual Hawaii Conf. on System Sciences, Vol. III, 1993 pp. 503-512.

[37] D. Marriott. Management Policy Specification, *Imperial College Department of Computing, Research Report DoC. 94/1,* Nov. 93, Available by FTP from dse.doc.ic.ac.uk.