Proceedings of the Conference on Organizational Computer Systems (COCS'91)
Atlanta, Georgia, 5–8 November 1991
SIGOIS Bulletin vol 12, nos 2 & 3, pp 171-184.

# The Representation of Policies as System Objects

*Jonathan D. Moffett & Morris S. Sloman*[1]

Imperial College of Science Technology and Medicine
Department of Computing, 180 Queen’s Gate, London SW7 2BZ, UK

## ABSTRACT

This is an exploratory paper in which we describe aspects of management policy which could be modelled as objects in a distributed computer system, in order to enable them to be queried and manipulated. Policies are 'the plans of an organisation to meet its goals'. They are persistent entities which are intended to influence actions, either by *motivating* actions or by *authorising* them. This distinction reflects the observation that agents only successfully carry out actions if they are both motivated and empowered to do so. In addition to persistence, policies have other main characteristics: they are directed to subjects; they are typically organised in hierarchies in which the goal of a policy is achieved by creating lower-level policies until identifiable actions are completed; and policies may conflict, so they require to have a precedence ordering.

There is a need to represent and manipulate policies, as objects within the computer system, so that they can be used to influence the activities of automated managers within large distributed computer systems. We describe a possible structure for policy objects and the operations which can be performed on them. Their attributes include: modality (positive or negative motivation or authorisation); policy subjects, goals, and target objects; and the constraints which may apply. The method of representation of relationships between policies is left as an open issue.

Related work and concepts in the modelling of policies are referred to, including a brief discussion of security models in this context. The open issues raised by this paper are described.

**Keywords**

Management policy, security models, authority, motivation, distributed system management.

[1] Email addresses: jdm@doc.ic.ac.uk & mss@doc.ic.ac.uk

## 1 INTRODUCTION

### 1.1 Policies in Distributed Systems

All formal organisations have policies, which are defined in the dictionary as 'the plans of an organisation to meet its goals'. They have two related purposes: to define the goals of the organisation; and to allocate the resources to achieve the goals. The policies are used as a means of management, in a hierarchical fashion. A high-level policy guides a manager, who may achieve its goals by making lower-level policies which apply to other managers lower in the hierarchy.

Most organisations issue Policy Statements, intended to guide their members in particular circumstances. Policies may provide positive guidance about the goals of the organisation 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 authorisation to) the resources which are needed to carry out the goals. If they allocate money they are typically called Budgets.

The background to this paper is work in Distributed System Management (DSM), particularly on the Conic [Magee 1989] and Domino [Law 1990] projects and in the Special Interest Group on DSM [Sloman 1987]. In this context, distributed systems are those in which several autonomous computers, exchanging information over a communications network, cooperate to achieve goals. It has become apparent that work on DSM inevitably involves discussion of policies which have to be agreed and set up by independent management agents in order to cooperate in distributed systems. Areas in which this is relevant include:

- Access control: authority cannot be delegated or imposed from one central point in a distributed system, but has to be negotiated between independent managers who wish to cooperate but who may have a very limited trust in each other [Moffett 1991].
- Configuration of systems: change management is governed by requirements for the maintenance of consistency which may be described by general principles which can be expressed as policies [Kramer 1989].
- Quality of Service (QoS) of communications, where end-to-end service policies for qualities such as reliability and security have to be negotiated and monitored [Sluman 1990].
- Accounting for communications across networks, where the need for integration of users' costs requires the coordination of policies [Estrin 1991].

A common theme in the above examples of DSM is the need for independent managers to be able to negotiate, establish, query and enforce policies which 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, then the PN Manager has 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 1a.

An additional policy has to be established (created) by the PN Manager to meet IC's requirements. One approach is to create a policy which motivates the PN Manager himself to generate the status information and provide it to the IC Network Manager regularly, as shown in figure 1b. An alternative approach to create a policy which gives the the IC Network Manager the authority to perform the operations needed to obtain the regular status information, as shown in figure 1c.

This example brings out one of the main points in the model. Policies which *motivate* activities and policies giving *authority* to carry out activities can each exist independent of each 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 her to do so.

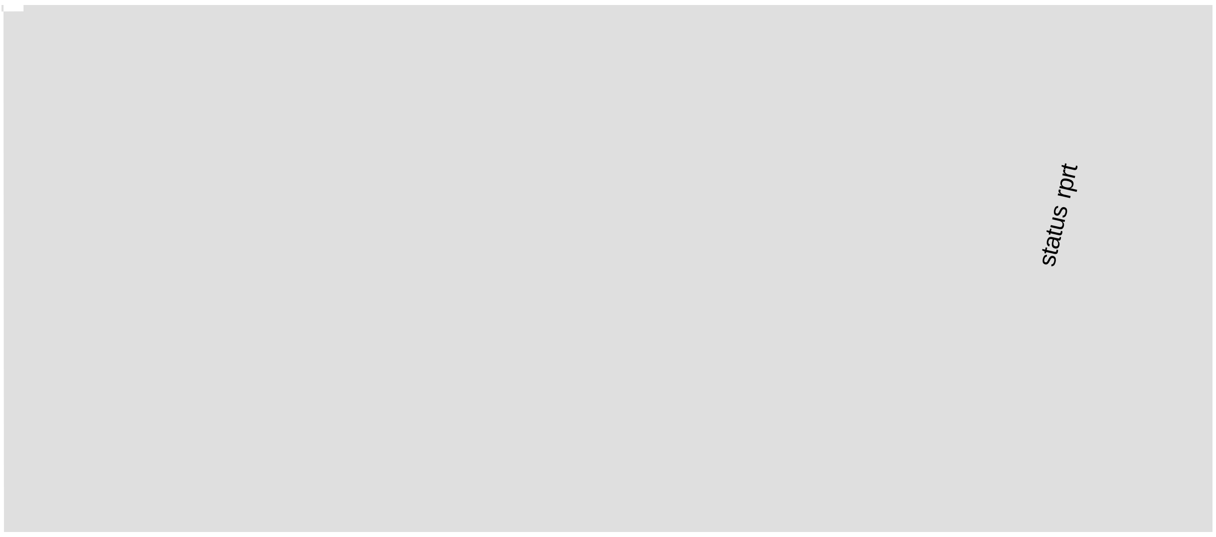

**Figure 1 Policies for PN & IC Managers**

## 1.2 The Motivation for Modelling 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 organisation's policies in a database permits staff to search, using keywords, for policies relevant to their proposed plans. The Pythagoras project [Bedford-Roberts 1991], discussed in section 6.2 below, is concerned with modelling policies for this purpose.

With the automation of many aspects of management in distributed systems and computer networks, there is the need to represent management policy within the computer system so that it can be interpreted by automated managers in order to influence their activities.

This paper owes its origin to the Management Policy Workshop held at Imperial College, London on 27th February 1991 [Sloman 1991]. The workshop explored the need for management policies in distributed systems, and how policies could be modelled as objects, in order to enable them to be queried and manipulated. This paper aims to create a framework capturing as many general aspects of management policies as possible, while exposing those issues which require further work.

## 1.3 Characteristics of Policies

We define some characteristics of the policies which we will be discussing in order to give a working definition which is more precise than simply ‘plans’. We start from the basis that policies are intended to influence actions. As shown in the example above, we distinguish between policies which are intended to motivate actions to take place and policies which give or withhold power for actions to take place. *Actions* are operations which are performed by agents provided two preconditions are satisfied: motivation and power (see figure 2):

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

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 unauthorised power. We take the simplifying view that all policies giving power can be viewed as giving *authority*, and in the rest of this paper we categorise policies as being either *authorisation* or *motivation* policies.

Agents are always humans, but it is convenient to extend the concept of 'action' to include computer processes which they have commanded to carry out actions. An agent's motivation is represented by him submitting a command to the computer system. His authorisation is represented by more than one mechanism: the access control system authorises his use of named resources, while an accounting system may authorise his use of commodities such as file store and cpu time, if their use is controlled.

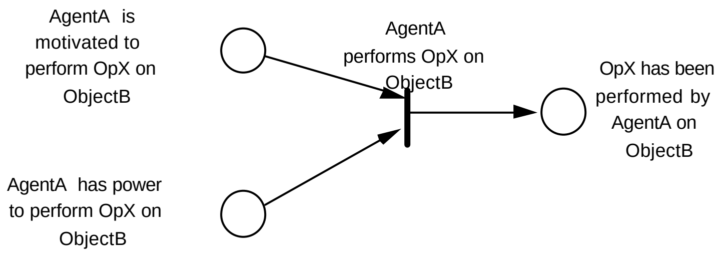
AgentA is motivated to perform OpX on ObjectB
AgentA performs OpX on ObjectB
OpX has been performed by AgentA on ObjectB
AgentA has power to perform OpX on ObjectB

**Figure 2 Preconditions for Action**

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 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. Ex1 is a command which is to be carried out immediately and has no persistence, so we do not regard it as a policy.

> The System Administrator is to back up department D's disc files to tape now. **Ex1**

Policies are about organisational goals, which need someone to achieve them. All our policies have *subjects*, the people to whom they are directed, although in some cases the policy applies to all members of an organisation. Ex2 has no agent to which it is directed, so we do not consider it to be a policy.

> It should always be possible to recover from media failure. **Ex2**

It is a fundamental characteristic of policies that they are organised in a hierarchical fashion. Examples Ex3, Ex4 and Ex5 illustrate this:

> The manager of Department D is to ensure that the department can always recover from media failure. **Ex3**

> The System Administrator is to back up department D's disc files to tape once a week. **Ex4**

> There is a system command (which was input by the System Administrator) to run a job which backs up department D's disc files to tape each Friday at midnight. **Ex5**

We may suppose that a director of an organisation has made policy Ex3 and given responsibility to the manager of department D to carry out the policy. The manager has made policy Ex4 and given responsibility to the System Administrator to carry it out, which he has done by creation of policy Ex5. We regard policies as being made at a high level by a manager and responsibility for achieving their goal being assigned to other members of staff, who may in turn do one of these things:

i) Create a lower level policy which will achieve the policy goal they have been assigned, and assign responsibility for it to another member of staff. This is what the manager did in the policy statement Ex4.

ii) Achieve the specified goal themselves by carrying out actions which achieve the goal, e.g. if the System Administrator manually runs a job each week to back up the files.

iii) Delegate the task not to another member of staff, but to a computer system which will carry out actions which achieve the the goal. This is what the System Administrator has done in Ex5.

It might be suggested that the distinction between authorisation and motivation policies is artificial. Since no action can be performed without the agent having both authority and motivation, there is no point in having separate types of policy. However, there are cases where we need to separate authorisation and motivation policies because it is sensible for a manager to have the authority for some action without immediate motivation to do it, or conversely to have responsibility, but no direct authority to do it. In the following examples Ex6 & Ex7 it will be simpler to describe the situation by having separate policies for authorisation and motivation:

X has authority to create access rules for the entire organisation. Other managers motivate him to create access rules according to criteria which they specify separately. **Ex6**

X is responsible for getting the office rewired, even though he is not an electrician and is not authorised to do rewiring. However, he has an adequate budget, and pays an electrician to do the job. **Ex7**

## 2 POLICIES AS OBJECTS

It is useful to view policies as objects which can be created, destroyed and queried. The object-based view is that an operation is performed by a user object (representing the agent) on a target object by sending a message to it. We extend this view to distinguish between an operation and and an *operation request*, which is a message issued by the user object which will only be delivered to the target object if it is authorised.

### 2.1 Components of Policy Objects

We model a policy, whether concerned with motivation or with authorisation, as an object having at least the following attributes:

- *Modality* - a policy has one of the following modalities: positive authorisation (*permitting*), negative authorisation (*forbidding*), 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.

  An example of a negative motivation policy is Ex8, similar in intention to Ex3:

  Department D is to prevent loss of data from media failure. **Ex8**

- *Policy subject* - This attribute defines the user objects to whom the policy *applies*, i.e who are authorised or motivated to carry out the policy goal within the limits defined by the policy constraints. The policy subject is an expression which defines a set of user objects or a predicate which a user object may satisfy. Where a policy has been automated as a computer system command we regard the user who inputs the system command as the policy subject. See section 2.2 below.
- *Policy target object*, which defines the objects at which the policy is *directed*. It also is an expression which defines a set of objects. See section 2.2 below.
- *Policy goal* - the goals or actions defined by the policy, which are modelled as *operations* to be performed on the target object. See 2.3 below.
- *Policy constraints* - predicates which must be satisfied before the policy is to have any effect. See 2.4 below.

In addition, we need a means of representing the relationships which can exist between policies: the *satisfaction* relationship between policies in a hierarchy; and the *precedence* relationship between conflicting policies, discussed in section 5, below. We do not know at this stage of research what is the best method of representing them.

## 2.2 Policy Subjects and Target Objects

Policy subjects and target objects may be specified either as a set of objects which can be enumerated or by means of a predicate which is to be satisfied. Example Ex3 above is an example of the former. The following examples, Ex9 and Ex10, show policies where the subjects and target objects, respectively, are defined in terms of predicates:

> [Any user who is] an Owner of an object may delegate Manager authority of the object to another user. **Ex9**

> All users must apply encryption to the transmission of any financial transaction which is over £20,000. **Ex10**

Policy subjects and target objects are not normally enumerated in terms of individual users as policy subjects or individual target objects. Typically a policy is expressed in terms of organisational positions, not individuals, and groups of objects.

One approach to specifying groups of objects and organisational positions is by using generic management domains [Sloman 1989]. They are objects which enumerate a set of objects to which a common management policy, such as access control, is to be applied.

One essential characteristic of management domains is that their membership can be evaluated at any time. Some policies, however, such as Ex9 and Ex10, are general principles which can be stated without necessarily being able to state at any moment the set of objects to which they apply. The predicate may be expressed in terms of an attribute such as financial value, or a role such as ownership.

## 2.3 Policy Goals

Policy *goals* may define either *high-level goals* or *actions*. Example Ex3 illustrates a high-level goal ' recover from [media failure]'; it does not prescribe the actions in detail, and we can imagine a number of different actions which could achieve the goal. On the other hand, example Ex5 illustrates an action 'run the BackUp job [on department D's disc 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, example Ex4, 'back up [department D's disc files]' might be regarded as an action policy, while in other situations it might be a goal in which the System Administrator has several options for action. Where there is no risk of ambiguity we abbreviate 'high-level goal' to 'goal'.

In order to distinguish between actions and goals, we need to have an alphabet of the operations which can be performed by objects in a system. Then any goal which 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 organisations' policy manuals, can be regarded as a sequence of actions.

Although examples Ex3 and Ex4 are motivation policies, the distinction between goals and actions is equally valid for authorisation policies. There may be a high-level authorisation policy, such as Ex11:

> No-one is authorised to carry out any financial transactions without specific authorisation. **Ex11**

We can translate this into our policy object model as 'All users (*subject*) are not authorised (*modality*) to carry out any financial transactions (*goal*) [on accounts (*target objects*)] without specific authorisation (*constraint*)'.

Actions represented by operations naturally have three components: the target object on which it is performed, the operation performed on it, 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 following example, Ex12, the amount of the transaction is a parameter of the operation, and forms a part of the authorisation policy:

> Financial managers are authorised to carry out financial transactions of a value up to £1 million. **Ex12**

## 2.4 Policy Constraints

The policy constraints component 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 authorisation 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, as shown in Ex13:

> Members of Payroll may Read Payroll Master files, *from terminals in the Payroll office, between 9 am and 5 pm, Monday to Friday*. **Ex13**

Policy constraints may be also be expressed in terms of conditions or attributes of objects such as users and target objects. See Ex14 and Ex15:

> John is responsible for backing up the department's files *until Jim returns.* **Ex14**

> *If a machine is identified as causing trouble to other users,* it will have to be disconnected from the network. **Ex15**

## 2.5 Operations on Policy Objects

For simplicity we assume a minimal set of operations which can be performed on policy objects:

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

We may or may not want to require authorisation to perform operations on policy objects. If the computer system is simply a documentation aid, we might want to have no restrictions on the operations at all, or perhaps just a warning if an invalid policy object is created (e.g. a filing clerk being the policy subject for the company's corporate strategy). 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 authorisation policies in [Moffett 1991]. A similar approach for motivation policies is discussed in section 4.2 below.

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, Ex16, 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. **Ex16**

The object which expresses that policy will typically be separate from the coding that implements it. On the other hand the system could use that policy object as part of its implementation; for example the policy could be represented 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 a persistent object will be to ensure that it is recognised 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

# 3 AUTHORISATION POLICIES

The most common form of authorisation policy is access control. We briefly consider security models of access control in section 3.1, and discuss one approach to discretionary access control in section 3.2.

However access control is concerned with named, atomic resources - objects. We have to consider also the power to use commodity resources, such as file store and cpu in computer systems, and commodity resources such as money and manpower when considering actions outside computer systems. This is an area for further study, not covered in this paper.

## 3.1 Security Models

Security models, in general, do not attempt to model policies as objects. However, policies are recognised as essential to security modelling. A number of different models of security policies are discussed in [Olson 1990].

The USA Department of Defense Trusted Computer System Evaluation Criteria (TCSEC) [DoD 1985] regard a security policy as a fundamental aspect of its approach. 'There must be [a] ... security policy enforced by the system. ... there must be a set of rules that are used by the system to determine whether ...'. The implication is that a policy is a set of rules which guide the actions of the system. We may note that, if a rule is viewed as a potential system object, this definition is compatible with our own view of policy objects. The definition of security policy in the OSI Security Architecture [ISO 1988] is similar: 'The set of criteria for the provision of security services'.

Logical access control is divided by the TCSEC into two categories: mandatory and discretionary. *Mandatory access control* enforces polices which are built into the design of the system and cannot be altered except by installing a new version of the system. TCSEC does not define it formally, but an example is the policy that in multi-layer security systems data cannot be read by a user with a lower security classification than has been assigned to the data. It defines *discretionary access control* mechanisms as those which allow users to specify and control sharing of objects with other users.

We need to consider how this categorisation fits into our model. There are in fact three different characteristics which appear to go into the Department of Defense definition of mandatory access control:

- The policy cannot be altered.
- The policy applies universally, i.e. the set of policy subjects is all the objects in the system.
- The policy has priority over all other policies, i.e. over discretionary access control policies.

With these implied assumptions, therefore, we can reconcile the two views of policies. But it must be recognised that the ability to model policies as objects is not very useful when discussing mandatory security.

## 3.2 Discretionary Access Control

We here discuss one particular approach to discretionary access control, covered in more detail in [Moffett 1990]. We define authorisation policy objects, *access rules*, whose purpose is to enable the system to decide whether to allow users to perform operations on target objects. A simple Access Rule is shown in Figure 3. It specifies a *User Domain*, which identifies the set of possible users who can perform operations; the *Target Domain*, which identifies the set of possible target objects on which operations can be performed, and the *Operation Set*, which are the authorised operations a user can perform on a target. There is no default access allowed to users. An operation request is authorised if and only if an access rule exists which applies to it, i.e. if the user object is in the set defined by the user domain of the rule, the target object is in the set defined by the target domain of the rule and the operation is in the operation set of the rule.

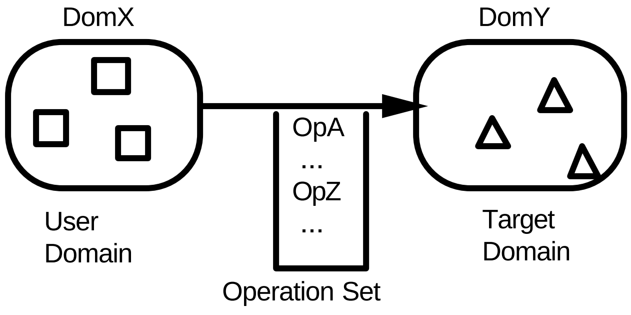

**Figure 3 An Access Rule**

We can see that an Access Rule maps quite precisely onto our policy model: the Modality is always positive authorisation; the Policy Subject is the User Domain of the Access Rule; the Policy Target Object is its Target

Domain; and the Policy Goal is its Operation Set. Constraints, typically relating to date, time or location of user, are also allowed in Access Rules.

## 4 MOTIVATION POLICIES

### 4.1 Responsibility

We have simply stated that a policy subject 'is responsible' for carrying out the goal of a motivation policy, and all that we have meant so far is that someone has the intention that the subject should carry out the action. For the moment we will simply assume that policy subjects do what is required of them, although this is of course too simple a view for adequate analysis of responsibility. However, we often talk about someone, who we call the *policy manager,* being ‘responsible to’ someone else. See the following example, Ex17:

> The Head of Department decides that the System Administrator shall be responsible to the Administration Manager for backing up the department’s files. **Ex17**

In order to represent this the policy object could be extended with a new attribute - the policy manager - to represent this situation, but we prefer to keep the policy object as simple as possible. Our preferred approach is to create another instance of a policy object, for which the subject is the policy manager, motivating him to take appropriate reward or punishment actions if the first policy is, or is not, carried out.

Ex17 is therefore represented by two policy objects, each created by the Head of Department. One motivates the System Administrator to do the backup actions, and report to the Administration Manager on their success or failure. The other motivates the Administration Manager to supervise the System Administrator and take appropriate action, depending on what the System Administrator does.

There is now the requirement that it is necessary to be able to refer to one policy from another. However, since this can be done by selection of a named attribute within an object, this does not give rise to any conceptual problem.

Note that, as discussed in section 1.3 above, we do not assume that, because a subject is responsible for actions, he has the authority to put his intentions into effect. He may have been given responsibility without power, which is a situation which may be deplored but must be able to be described. Alternatively, the power may be indirect, as in the example Ex9.

### 4.2 Operations on Motivation Policy Objects

The operations on motivation policy objects are the straightforward ones of Create, Destroy and Query. If the policy objects are regarded as more than a documentation aid, then we want to support the notion that users can only create policy objects within the scope of their responsibility. The concept of Delegation of Responsibility is a familiar one, and in this model will be represented by the creation of motivation policy objects within the constraints of a hierarchy of managerial responsibility. In this section we do no more than sketch out a possible approach.

We represent a manager’s delegation of responsibility for a goal to another user agent as the creation of a new policy object, as we have seen above. The Creation operation is itself subject to a fixed policy.

The fixed policy for creation of a new motivation policy which we recommend is that he should be able to create the new policy object only if three conditions are met:

- The set of users to whom responsibility is delegated (the subject of the new policy object) is a subset of those which he can actually motivate to perform the goal. We will say that the set of users is within the *user scope* of the manager.
- The set of target objects to which the new policy object is directed is a subset of those on which he is motivated to achieve his own goal. We will say that the set of target objects is within the *target scope* of the manager.

- The goal of the new policy object is a goal which the manager is motivated to achieve by some existing policy. We will say that the goal is within the *goal scope* of the manager.

The user and target scopes needed are identical to those which are used in the delegation of authority - a domain expression which defines a set of users. See [Moffett 1991] for a discussion of this. On the other hand, definition of ‘goal scope’ is impossible until we have successfully defined goals. However, the intuition is that goals should be ordered, and that policy creation should only be permitted if the goal in the created policy is less (in the ordering) than the goal scope of the manager. As an example, *selling* may be considered ‘less’ than *marketing*, which includes market research and product development as well as selling. This is an area which requires further work.

## 5 INTERACTIONS BETWEEN POLICIES

### 5.1 Policies as Collections of Policy Statements

[Holden 1991] does not attempt to present a complete model of policies, but concentrates on (motivation) policies as collections of statements. He distinguishes between *action*, *goal* (high-level goal in our usage) and *rule* (negative motivation in our usage) statements. From some examples taken from system management, he makes a number of observations:

- Policies are often not derived from single policy statements, but from a collection of statements. Goal statements are often constrained by one or more rule statements.
- Policies can interact and sometimes conflict. The result can be undecidable or self-contradictory.
- Goal statements are often under-specified, and may become fully specified through: other policies; external constraints, e.g. physical or legal; and/or choices made by the management system.
- Policy statements assume the existence of information to resolve references, e.g. a quantitative measure of quality of service or knowledge of the structure of a system.

### 5.2 Hierarchical Policies

Hierarchical policies were introduced in examples Ex3 – Ex5 in section 1.3 above. Two questions arise from the informal example:

- What does it mean for one policy to ‘achieve the policy goal’ of another?
- What are the relationships between creators and subjects in a hierarchy of policies?

We do not attempt to answer these questions in this paper. They require further work.

### 5.3 Ordering of Policies

There needs to be a precedence ordering policies in two circumstances: when policies conflict; and when two actions are incompatible, as when there is competition for the allocation of scarce resources, needing policies for priority.

There are a number of ways in which policies can conflict. The simplest is when there are two policies, one of which motivates or authorises a subject to perform an operation on an object and the other deters or forbids it. Other conflicts are more complex, e.g. when an overriding 'separation of duties' policy declares defined pairs of actions to be in conflict. Simple conflicting policies need a precedence ordering to determine which is to have priority. For example, there is an assumption in existing security models that mandatory security policies have precedence over discretionary security policies. If a discretionary policy authorises an action and a mandatory policy forbids it, it is forbidden.

Policies defining priorities state an explicit precedence order, e.g. example Ex18

The following priority ranking will be given to requests for [purchase of] hardware or software resources: 1) Mainline teaching resources ....2) Equipment for support staff ....3) Equipment for optional courses .... **Ex18**

We interpret a set of priorities as interacting policies; the policy subject is motivated to perform the first priority action, unless there is no longer any requirement for it, and then the second one, and so on. Priorities require another concept, *satisfaction* of a policy, after which policies with a lower priority may be addressed.

## 6 OTHER RELATED WORK

The related work in this area can be characterised in two orthogonal dimensions: motivation and authorisation policies; and policies as *objects* (in some very broad sense) or as *social processes*. The work on *contract models*, *Pythagoras* and policies as *collections of statements*, deals with policies as motivating objects, and the *process approach* also concentrates on motivation. The work on *security models*, in particular on *delegation of authority*, discussed in section 3 above, deals with authorisation policies.

### 6.1 Contract Models

The approach in ISTAR [Dowson 1987], an Integrated Project Support Environment is to use contracts as the model for activities in the software development process. Each activity is conducted by a 'contractor' (e.g. a programmer), for a 'client' (e.g. a manager). It has precisely defined deliverables and acceptance criteria, and other contractual conditions. Where the size or complexity of a contract warrants, the contractor is free to issue 'subcontracts' to help fulfil the original contract; the subcontractors may themselves issue 'subcontracts' and so on. The collection of tasks that compose a complete software project forms a *contract hierarchy*. At the root of this hierarchy is the contract for the project as a whole. The leaves of the hierarchy are the self-contained contracts which are completed without letting subcontracts. The intermediate nodes of the hierarchy are subcontracts which themselves let subcontracts. Eventually all the subcontractors will complete their assigned tasks, allowing completion of the original contract.

The TOBIAS project [Marshall 1991] follows this approach, and introduces *responsibility* as the central element of a contract. A *role* specifies a set of types and possibly a subset of their operations. *Agents* are responsible for carrying out their assigned roles, and are responsible to their clients as their source of authorisation. Contracts specify the responsibilities of agents by setting out the roles they can play and for whom. The application example is different in kind from ISTAR. ISTAR sees contracts as being created for the performance of specific development tasks, and discharged on completion. The example in TOBIAS, on the other hand, is relationships in the UK National Health Service between patients, doctors and management and regulatory agencies. These, and therefore the contracts between them, are inherently persistent, and should be regarded as policies as we have defined them here.

One approach to cooperation in distributed problem-solving [Smith 1981] - *task-sharing* -uses contracts. In this situation a contract is an explicit agreement between a node that generates a task (the manager) and a node willing to execute the task (the contractor). The manager is responsible for monitoring the execution of a task and processing the results of its execution. The contractor is responsible for the actual execution of the task. A contract is established by a multi-stage process of mutual selection: advertisement by the manager; bid submission by available nodes; bid evaluation by the manager; and conclusion of the contract between the manager and the successful bidder.

### 6.2 Pythagoras

The Pythagoras project [Bedford-Roberts 1991], is concerned with modelling policies in order to create a database of the policies of an organisation. Initially its purpose is 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, although this might be considered as a future development.

A *policy* in Pythagoras is a right or responsibility declared so as to prompt action conformant with an intention. There are two types of occasion when it is normal to apply policy: firstly, when intention and consequent action are separated in time; and secondly, when intention and consequent action are associated with different people. This is consistent with our view of policies as being essentially persistent, because the action cannot be an immediate consequence of an intention.

There is another interesting distinction in Pythagoras, between *satiable* and *insatiable goals*, which could respectively be viewed as: goals which are achieved, like ISTAR contracts, by carrying out actions; and goals to maintain state, e.g. ‘keep the department running’.

## 6.4 Deontic Logic

Deontic logic, because of its ability to specify the concepts of obligation and permission directly, is an attractive candidate for expressing policies. The natural approach is to equate motivation with obligation, and authority with permission, and thus model policies in terms of existing logical models.

The work apparently closest to our interests is [Lee 1988], which proposes the use of deontic logic for the description of bureaucratic systems and authority hierarchies. He uses a language similar to Prolog to record the deontic status of users and actions. Although the language allows expression of obligation as well as permission, the examples are all confined to permission, and the relationship between obligation and permission is not explored.

Modal Action Logic (MAL) [Jeremaes 1987] provides a means of system specification using modal logic. Obligation (*obl*) expresses the notion of liveness, that an action must happen at some time in the future, while permission (*per*) expresses the notion of safety, that an action is permitted to happen. However, this version of MAL is unsatisfactory for our purpose, because *obl* overrides *per* in MAL; one cannot express authorisation and motivation policies independently. [Fiadeiro 1990] removes the overriding of *per* by *obl*. However, the intuitive view of motivation is a long way from *obl*. '.. a live trajectory is such that every event that is obligatory after a prefix of the trajectory will occur later on.' Although this captures the notion of liveness, it is too strong for our purposes. What we mean when we say someone is motivated to do something is something such as 'he will do this action at some point in the future, unless another event occurs which removes his motivation'. In other words motivation may cease to be true, because the policy has changed before he has been able to carry out the action, whereas *obl* appears to be perpetual. These comments on MAL should not be interpreted as criticism of it for the purpose for which it has been designed: embedded real-time systems. However, it does illustrate that it cannot be extended simply to our application.

There appear, to the potential consumer of deontic logic, to be three main aspects to the requirements which affect the design of the logic:

a) The orthogonality of the two main modes of a policy, motivation and authorisation.

b) The need for policies to be able to be created and then destroyed without residual effects.

c) The need to be able to resolve conflict between policies.

Aspect a) implies a more complicated set of modalities than is supplied by the work mentioned above. The work of Pörn on action theory [Pörn 1977] provides this. We are examining the possibility of equating our ‘authorisation’ to the modality M, where $M_a p$ says ‘it is possible for a that p’. ‘Motivation’ might then be represented by his ‘intentional action’, which says that ‘a intends to bring it about that q’. Aspect b) does not appear to present major problems, as it does not present any demands beyond those already provided by an action logic such as MAL.

The possibility of conflicting policies - aspect c) - means that we cannot take a straightforward view of any single policy statement. [Wieringa 1989] refers to two ways of interpreting ‘It is forbidden to park here,’ as the *promulgation* of a rule or as the *observation* that a rule exists. Similarly the statement in Ex6 that ‘Payroll clerks have authority to Read Payroll files’ may be a correct observation of a policy, but is not necessarily the promulgation of a rule, e.g. if there is another policy in existence, with higher priority, which denies that authority. Similar comments apply to motivation polices where a positive motivation is frustrated by the combination of limited resources and another policy with higher priority. Our way ahead will either be to treat policies as statements in this way, or else to treat them as defaults which may be overridden.

Our work on using deontic logic for policy specification is at an early stage, and work is continuing.

## 7 SUMMARY & CONCLUSIONS

### 7.1 Summary

We here summarise the concepts used in this framework.

We model a *policy* as a persistent object with a unique identifier. A policy object has four attributes:

- *Modality* - possible values are: *motivation* (positive or negative); *authorisation* (positive or negative). A motivation policy is one which influences whether the user attempts to achieve a goal. An authorisation policy is one which influences whether the user has the power to achieve it. Both are required for successful achievement of a goal.
- *Policy subject* - a set of user objects to whom the policy applies. This may be defined by enumeration, e.g. by the use of domains, or by predicate.
- *Policy target* - a set of objects to which the policy is directed.
- *Policy goal* - the goals or actions defined by the policy.
- *Policy constraints* - predicates which must be satisfied before the policy is to have any effect.

Policy *goals* may define either *high-level goals* or *actions*. High-level goals do not prescribe specified operations, and a number of different actions may achieve a high-level goal. Any goal which is expressed purely in terms of the alphabet of the operations known in a system is an action policy, and any other goal is regarded as a high-level goal.

We can analyse an action into three components: the *object* on which it is performed, the *operation* performed on it, and one or more *parameters* to the operation. A similar analysis for high-level goals is thought to be possible but is not attempted in this paper.

Policy *constraints* are predicates which may be expressed in terms of general system properties, such as extent or duration, or some other condition. They determine whether a policy is to be applied in a particular context.

Some policies may be altered dynamically while others are *fixed policies*, fixed for the life of the system. The minimal set of operations which can be performed on dynamically alterable policy objects is *Create* , *Destroy* and *Query*. Users need authority for these operations in the same way as for any other operations on objects. We may require to place additional controls on the performance of operations on policy objects, to ensure that they are within the scope of the user who performs them.

*Authorisation policies* may be partially implemented as *access control* systems whose purpose is to enable the system to decide whether to allow users to perform operations on objects. However, authorisation policies for use of *commodities* are outside the scope of access control, but they also need to be modelled.

*Motivation policies* are often associated with *responsibility*. We do not attempt to define responsibility, but observe that when an agent is responsible to a manager for achieving a goal, two motivation policies are typically required, one to motivate the user to achieve the goal, and one to motivate the manager to supervise the user.

We have identified three kinds of interactions between policies:

- Hierarchical policies;
- Conflicting policies relating to a single action;
- Policies which define the relative priority of actions.

### 7.2 Outstanding Issues

We have identified a number of open issues in this paper.

**Hierarchical Policies** - the hierarchical relationship of policies is fundamental to most views of them. This requires the means of specifying hierarchical relationships between policy objects, of specifying high-level policy goals, and defining when completion of one goal or action achieves another goal.

**Precedence Ordering of Policies** - This requires the means to enable the resolution of conflicts between policies and the expression of policy priorities.

**Reasoning about Policies** - since a number of different policies may potentially apply to a single action, we need to be able to reason about the effects of multiple policies. The possibility of using deontic logic for this purpose requires exploration.

**Implementation Tools** - this paper has not addressed implementation issues at all. Tools are needed for the representation and interpretation of policies in automated systems. Although these exist for special cases of authorisation policies in the form of access control systems, they also need development for motivation policies, and generalisation so that users have a uniform view of the two kinds of policy.

Other issues are: the specification of policy constraints; the development of authorisation policies for commodities; and the scope ordering of goals and actions (to control creation of motivation policies).

### 7.3 Conclusion

This paper has explored the issues relating to the representation of management policies so that a framework for generic protocols for the definition of management policies can be established. It has distinguished two main types of policy, relating to motivation and authorisation, and identified a number of characteristics which are common to most policies. It has put forward a view about how they can be modelled as objects. This can form the framework of a model of policies which will enable the construction of protocols so that independent managers can negotiate to create policies which will enable them to work cooperatively.

The aim of our analysis has been to delineate the scope of the subject and expose the issues to be solved, particularly on relationships between different policies. Our work is at a very early stage and we recognise that we do not have the solutions to the problems of representing management policy. Our intention has been to open discussion and stimulate further work in the area to build and improve upon the initial framework we have suggested.

## ACKNOWLEDGEMENTS

We acknowledge the contribution of the attendees at the the Management Policy Workshop in stimulating the ideas of this paper, in particular James Bedford-Roberts, whose constructive comments during extensive email correspondence were most helpful. This work was carried out with the support of the UK DTI/SERC (Grant No. GR/F 35197) for the Domino project.

## REFERENCES

[Bedford-Roberts 1991] Bedford-Roberts J., Concepts from Pythagoras, report HPL-91-22, February 1991, Hewlett-Packard Laboratories, Bristol, UK.

[DoD 1985] Department of Defense (USA), Department of Defense Trusted Computer System Evaluation Criteria, DOD 5200.78 - STD (Dec 1985), (microfiche).

[Dowson 1987] Dowson M., ISTAR - An Integrated Project Support Environment, Proc. 2nd Sigsoft / Sigplan Symposium on Practical Software Development Environments, Palo Alto, CA, Dec 1986, J SIGPLAN Notices, Vol 22 no 1 (January 1987), pp 27 - 33.

[Estrin 1991] Estrin D. & Zhang L., Design Considerations for Usage Accounting and Feedback in Internetworks, to be presented at IFIP Symposium on Integrated Network Management, Washington, USA, April 1991.

[Fiadeiro 1990] Fiadeiro J. & Maibaum T.S.E., Describing, Structuring and Implementing Objects, REX Workshop on Foundations of Object-Oriented Languages, May 1990.

[Holden 1991] Holden D.B., An Exploration of the Nature of Management Policy, ESPRIT/5165/harw/T2.1/1_0, AEA Industrial Technology, Harwell Laboratory, Oxfordshire, UK, 5 Feb 1991.

[ISO 1988] ISO, Open Systems Interconnection: Security Architecture, ISO 7498/2, 1988.

[Jeremaes 1987] Jeremaes P., Khosla S. & Maibaum T.S.E., A Modal (Action) Logic for Requirements Specification, in Brown 1986, pp 278-294.

[Kramer 1989] Kramer J., Magee J. & Young A., A Refined Model of Change Management in, Distributed Systems, 3rd Workshop on Large Grain Parallelism, SEI/CMU Pittsburgh October 1989.

[Law 1990] Law A.D., Sloman M.S. & Moffett J.D., The 'Domino' Project, Data Management '90 Conference, Egham, UK, 2-3 April 1990, BCS Data Management Specialist Group, pp 143-154.

[Lee 1988] Lee R.M., Bureaucracies as Deontic Systems, ACM Transactions on Office Information Systems, vol 6, no 2, April 1988, pp 87-108.

[Magee 1989] Magee J., Kramer J. & Sloman M., Constructing Distributed Systems in Conic, IEEE Transactions on Software Engineering, vol 15 no 6, June 1989, pp 663-675.

[Marshall 1991] Marshall L., Contracts for Controlling Management, in [Sloman 1991].

[Moffett 1990] Moffett J.D. Sloman M.S. & Twidle K.P., Specifying Discretionary Access Control Policy for Distributed Systems, Computer Communications, vol 13 no 9, pp 571-580 (November 1990).

[Moffett 1991] Moffett J.D. & Sloman M.S., Delegation of Authority, in I. Krishnan & W. Zimmer (eds), Integrated Network Management II, North Holland (1991), pp 595-606.

[Olson 1990] Olson I.M. & Abrams M.D., Computer Access Control Policy Choices, Computers & Security, vol 9 (1990) pp 699-714.

[Pörn 1977] Pörn I., Action Theory and Social Science, D. Reidel Publishing Company, Dordrecht, 1977.

[Sloman 1987] Sloman M.S. (ed), Distributed System Management, in Issues in LAN Management, ed I. Dallas & E. Spratt, pp 15-46 North Holland 1988.

[Sloman 1989] Sloman M.S. & Moffett J.D., Domain Management for Distributed Systems, in Meandzija & Westcott (eds), Proc of the IFIP Symposium on Integrated Network Management, Boston, USA, May 1989, North Holland, pp 505-516.

[Sloman 1991] Notes from the Management Policy Workshop (chairman, M.S. Sloman) held at the Department of Computing, Imperial College, London, 27th February 1991.

[Sluman 1990] Sluman C., Domino QoS Framework, Domino paper C1/Sema/3.1 July 1990, available from Dept of Computing, Imperial College, London.

[Smith 1981] Smith R.G. & Davis R., Frameworks for Cooperation in Distributed Problem, Solving, IEEE Transactions on Systems, Man & Cybernetics, Vol SMC-11, no 1 (Jan 1981), pp 61-70.

[Wieringa 1989] Wieringa R., Meyer J-J. & Weigand H., Specifying Dynamic and Deontic Integrity Constraints, Data & Knowledge Engineering 4 (1989), North-Holland, pp 157-189.