IEEE Journal on Selected Areas in Communications, 1993. vol 11(9): pp 1404-1414

Policy Hierarchies for Distributed Systems Management

5 July 1993

Jonathan D. Moffett

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

Morris S. Sloman

Imperial College of Science Technology and Medicine
Department of Computing
180 Queen's Gate
London SW7 2BZ
Email: mss@doc.ic.ac.uk

Abstract

Distributed system management, involves monitoring the activity of a system, making management decisions and performing control actions to modify the behaviour of the system. Most of the research on management has concentrated on management mechanisms related to Network Management or Operating Systems. However, in order to automate the management of very large distributed systems, it is necessary to be able to represent and manipulate management policy within the system. These objectives are typically set out in the form of general policies which require detailed interpretation by the system managers. This paper explores the refinement of general high-level policies into a number of more specific policies to form a policy hierarchy in which each policy in the hierarchy represents, to its maker, his plans to meet his objectives and, to its subject, the objectives which he must plan to meet.

Management action policies are introduced, and the distinction between imperatival and authority policies is made. The relationship of hierarchies of imperatival policies to responsibility, and to authority policies, is discussed. An outline approach to the provision of automated support for the analysis of policy hierarchies is provided, by means of a more formal definition of policy hierarchy refinement relationships in Prolog.

Keywords:
Management policy, policy specifications, authorisation, obligation.

1. INTRODUCTION

Distributed system management, the task of maintaining the service required of a system, involves monitoring the activity of a system, making management decisions and performing control actions to modify the behaviour of the system (Sloman & Moffett 1989). Management policies guide the decision making process; the manager has to interpret policies in order to achieve the overall objectives of the organisation. We define policies as the plans of an organisation to achieve its objectives.

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 policy is coded into manager components they become inflexible and cannot be reused in different environments. There is thus a need to be able to specify, represent and manipulate management policies without building them into manager components. This permits manager components to be reused in different environments and to be provided with the specific policies for each. Separation between manager components and the policies which they interpret enables dynamic changing of management policies for a system without changing the manager components.

A distinction is often made between policies and plans; typically plans are described as the means by which policies are implemented. We make no such distinction; our working definition of policies as 'the plans of an organisation to meet its objectives' deliberately compounds them. However, the definition does recognise these two aspects. The same policy represents, to its maker, his *plans* to meet his objectives and, to its subject, the *objectives* which he must plan to meet. Every intermediate policy in a hierarchy has these two aspects; viewed from above it is a plan, and viewed from below it is an objective.

This paper is the third in a series which proposes a model for the representation of management polices in distributed systems and explores how the model may be applied to some main concerns in making policies. The first (Moffett & Sloman 1991b) proposed the model and suggested a number of research avenues. The second (Moffett 1993) set out specifically to explore how policy conflicts could be analysed and resolved within the model. The current paper is a discussion paper which aims to explore the refinement of high-level policies into a number of more specific policies to form a **policy hierarchy**.

The structure of this paper is as follows. In section 2 we introduce management action policies and give an example of a policy hierarchy. Section 2.3 explains the reasons for wishing to analyse policy hierarchies. Section 3 provides a discussion of **responsibility** and its relationship to policy hierarchies, section 3.1 giving an analysis of the different kinds of policy which may be involved in responsibility, and section 3.2 discussing the acceptance of responsibility. Section 4 discusses the relationship between imperatival and authority policies. Section 5 provides an outline of an approach to the provision of automated support for the analysis of policy hierarchies by means of a more formal definition of refinement relationships in Prolog.

2 MANAGEMENT ACTION POLICIES

2.1. Policy Representation

We briefly summarise the generic characteristics of a **management action policy** which we represent as an object which can be manipulated by performing operations on it. More details can be found in (Moffett & Sloman 1991b). We do not claim that all policies can usefully be represented this way; we refer below to policies about management action policies (PAMAPs), and there are no doubt many other kinds also. However, management action policies prove to be useful for discussing the tasks of distributed system management, and the single word 'policy' should be assumed to refer to them in this paper.

Policies are intended to influence actions, but are not instant decisions to perform a one-off action, which is then executed. If a manager specifies that something is to be done once only and instantly, he does not need to create a policy; instead he can simply cause the action to be carried out by telling someone else to do it or

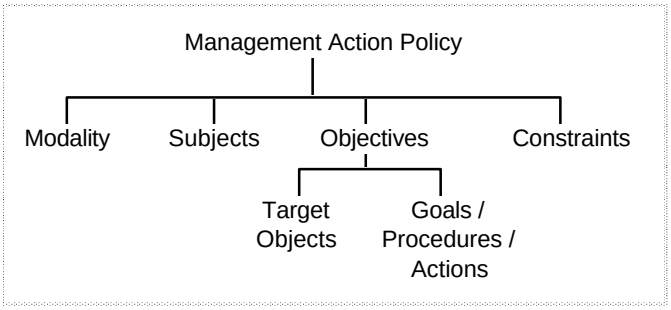

Figure 1 Management Action Policy Attributes

invoking a management operation on a managed object. A policy is a *persistent* specification of an objective to be achieved or a set of actions to be performed in the future or as an on-going regular activity.

A generic management action policy has the attributes shown in figure 1

Modality

We have identified two different classes of policies.

i) **Imperatival** policies give an agent the imperative to carry out an action provided he has the power to do so. In most cases the imperative is one of **obligation**, and in this paper we extensively use "obligation" as an example of imperative.

ii) **Authority** polices provide an agent with the legitimate power to perform an action, i.e. the operations an agent is permitted to perform on a target object. Access rules as examples of authority policy are explained in (Moffett, Sloman & Twidle 1990).

These two classes can be further divided into positive imperatival (obliging or requiring), negative imperatival (inhibiting), positive authority (permitting), and negative authority (prohibiting).

Hierarchies of the kind which we are discussing are concerned with imperatival policies, and the modality is assumed to be 'imperatival' throughout the rest of this paper unless stated otherwise.

Subject

The policy **subjects** are the entities to whom the policy is directed, i.e. who are authorised or obliged to carry out the policy objective. The subjects may be human or automated. Policy subjects are typically specified as sets, because a policy is typically expressed in terms of organisational positions and domains of objects, not individuals. One approach to specifying organisational positions and enumerating groups of objects is by using management domains (see below).

However, we have already remarked when discussing policy conflicts in (Moffett 1993), that if there is more than one individual manager who is subject of a imperatival policy then there is the possibility of conflict as a result of multiple managers attempting to achieve the same objective. Typically, automated or human subjects require a coordination protocol to prevent conflict, e.g. a token passing or election protocol to ensure that only one subject is active at a time. We therefore refer to a 'subject' in the singular. A methodical approach to the problem of avoiding conflicts between multiple policy subjects requires further research.

Target Objects

The policy **target objects** are the objects at which the policy is directed. In a large distributed systems environment it is impractical to specify separate policies for individual targets. For example the same policy may apply to all people in a department or to all of the set of files pertaining to an application. Like policy subjects they are typically defined in terms of sets, which may be specified either by enumeration or by

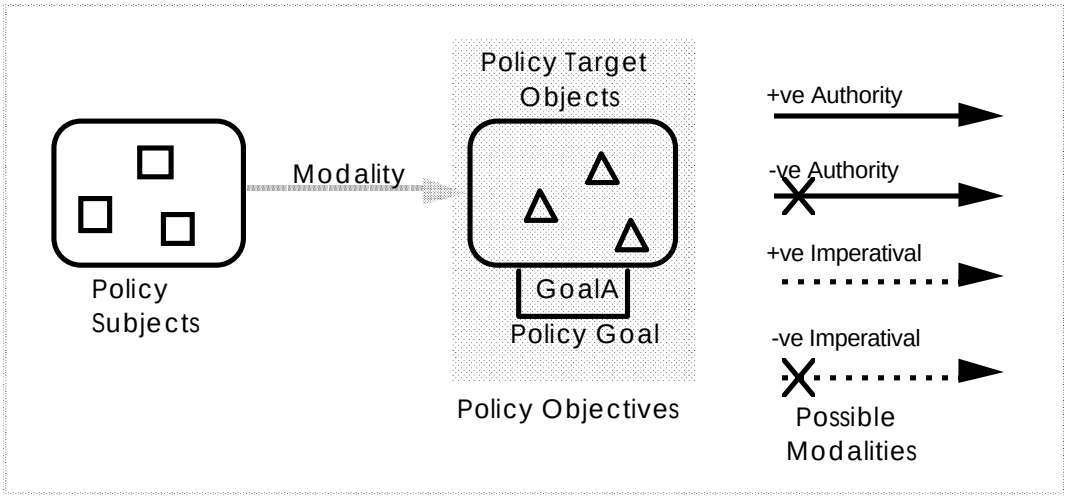

Figure 2 A Management Action Policy

means of a predicate which is to be satisfied. We have introduced the concept of enumeration by **management domains** as a means of grouping objects to which a common policy applies (Sloman & Moffett 1989, Moffett & Sloman 1993).

Policy Objectives & Goals

We use the term **policy objectives** for the aims of a policy. Objectives are expressed as a pair of **goals** and target objects. The policy goal defines either a **high-level goal** or a **procedure.** Procedures are a defined sequence of **actions**. Any goal which is expressed purely in terms of an alphabet of operations on objects in a system (i.e. a defined sequence of actions), is a procedure, otherwise it is to be regarded as a high-level goal, which requires refinement in order to define the actions which will achieve it.

An example of a high-level goal in an objective is '*protect* files from loss due to fire or media failure'. It does not prescribe the actions in detail, and there are at least two different procedures which could achieve the goal:

- *Keep* files in a fire-protected area and *use* very reliable processing equipment. This will reduce the likelihood of loss.[1]
- *Take backup copies* of the files, *store* them elsewhere and use previously defined recovery procedures in the event of failure. This does not reduce the likelihood of loss of the originals, but means that their recovery is possible.

By contrast, it is a requirement of procedures and actions that they should be well defined and unambiguous. An example of an action is '*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, the goal of '*back up* department D's disc files' might be regarded as an action, while in other situations it might be a high-level goal in which the System Administrator has several options for action.

The DOMAINS project (1991) distinguishes between:

- Goal - high level specification of objectives;

---

[1] This procedure would in practice need further refinement, so as to be expressed in terms of well-defined actions.

- Policy - a general statement about how a manager will achieve management goals;
- Plan - a procedure of actions, derived from policies, which can be deterministically evaluated at the time it is to be executed.

Constraints

These are a set of predicates which constrain the applicability of the policy. They may be expressed in terms of general system properties, such as extent or duration, or some other condition. An example of constraints in authority 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.

Graphical Notation

We use a standard graphical convention for illustrating policies (omitting constraints), as shown in figure 2. We use the convention that managers are represented by squares and other objects by triangles.

2.2 Policy Hierarchy

A high-level policy may be used as the basis from which multiple lower-level policies are derived. The derivation can be by refining the goals, partitioning the targets, or delegating the responsibility to other managers. Figure 3 shows an example of a hierarchical set of policies relating to protection of a company's assets.

We have identified several different relationships which may exist between policies in a hierarchy, as explained below.

Partitioned Targets

The target set of the lower-level policy may be a subset of the target set of the higher-level one, while the goal is the same. For example the target of policy 2 (Dept D's Assets) is a subset of the target of policy 1 (Company X's Assets). It is assumed there are similar policies for Departments A, B and C etc. so that all of Company X's Assets are protected.

In general there is a need to ensure that the whole target is 'covered' by lower-level policies in order to ensure that the partitioning is complete.

Goal Refinement

The goal of a high-level policy may be refined into one or more lower-level goals, referring to the same target; policy 4a has the same target as policy 3a (D's Files), but the goal has been refined from 'Protect from Loss' to 'Backup Weekly'.

There can be no automatic method of goal refinement - it is a matter of real-world judgement to decide what refines a goal and what is required for its complete fulfilment. One manager may decide that the weekly back-up procedure is adequate, while another may require a monthly inspection of the back-up store contents also.

The example specifies all higher-level goals in the form of actions, e.g. 'protect'. An alternative is to express the goal in terms of maintenance of a 'protected' state. The actions which refine the goal are the same in either case.

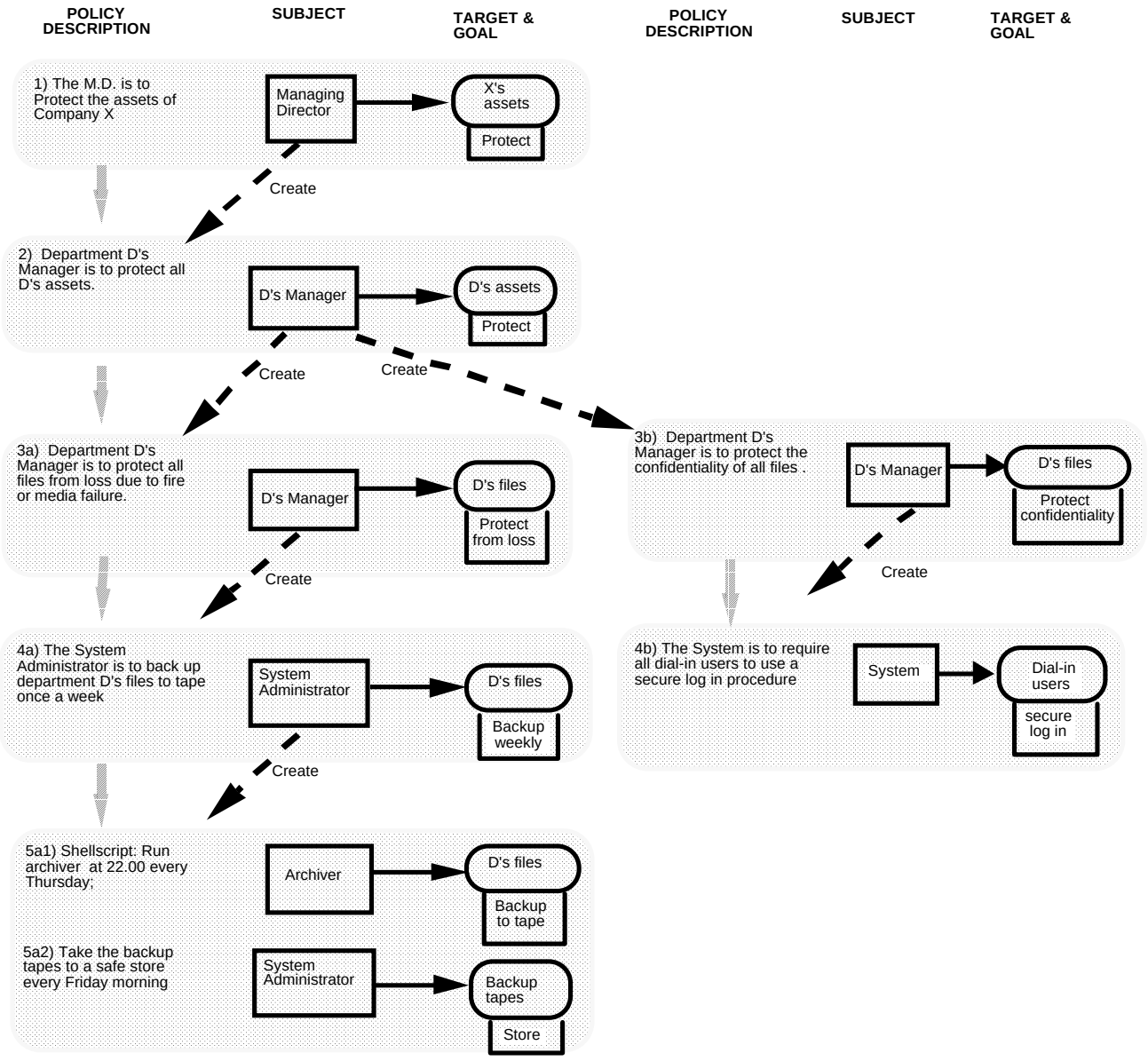

Figure 3 A Policy Hierarchy

Arbitrary Refinement of Objectives

There is form of refinement of objectives in which the goal and target are quite different from the higher-level objectives . For example, in refining policy 3b (to protect the confidentiality of all files) to policy 4b (all dial-in users must use a secure login procedure), there is no direct relationship between the subjects or targets of the two policies. However 4b is one of the policies needed to achieve the goals of 3b.

Procedures

A policy may be refined by an unordered set of lower-level ones, e.g. 'Protect from Loss' and 'Protect Confidentiality', and/or by a procedure, which is an ordered sequence of actions, e.g. 'Back-up to Tape' *and then* 'Take to Safe Store'.

Delegation of Responsibility

In most, but not all, of the cases in the example, the subject of the lower-level policy is different from that of the higher-level one. One subject **delegates responsibility** for the objective to another subject. In all the cases in the example the objective has been refined at the same time, but this need not be the case. Yemeni et al (1991) describe a mechanism for delegation which enables a manager to transfer the responsibility for performing management functions to a remote agent.

2.3 Reasons for Policy Hierarchies

The main 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. This may be possible for partitioned targets where lower-level policies can be expressed as subset relationships of higher-level ones. However it becomes increasingly difficult with goal refinement and in most cases impossible for objectives which have an arbitrary relationship.

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 becomes increasingly difficult for goal refinement and arbitrary relationships.

If the managers were all human beings, such an approach might be unnecessary. However, more and more of the entities which we call a 'manager' in a distributed system are not human beings, but automated processes which carry out policies. There are clear potential gains in efficiency and flexibility if these automated managers can be presented with relatively high-level policies to guide them, and information on the lower-level actions which are considered to meet these policies. In other words, they need precisely formulated policy hierarchies to drive them.

There is therefore a need to be able to describe formally the relationships between high-level policies, refined lower-level policies, and the mechanisms, actions and procedures which finally implement them. These policy hierarchies need to be constructed in such a way that a human manager can determine whether an organisation's policies are being satisfied.

Since, as discussed above, there can be no automatic way of refining policies without human intervention, there needs to be a means of codifying the refinement steps and, as far as possible, generalising them. This is clearly a possible application for an expert system, which could be used both for enabling independent human managers to evaluate the policies of other domains, and as a driver for automated managers. A possible approach to provision of automated support for this task is introduced in section 5.

Note that so far as human subjects are concerned, it is not critical whether the higher-level subject is actually the manager (in the sense of 'the boss') of the lower-level subject; the System Administrator may belong to a central administrative department, or even be a contractor, rather than being an employee of department D. What is critical is that there should be some means of effectively making the lower-level subject accept an imperative to carry out the policy.

2.4 Related Work

Masullo and Callo (1993) have specified six levels within a policy hierarchy:

- Societal Policy (Principles) such as ethics or laws;
- Directional Policies (Goals) such as organizational or corporate goals;
- Organizational Policy (Practices) such as contractual agreements or quality programs;

- Functional Policy (Targets) such as workload targets or quality measures;

- Process Policy (Guidelines) such as automated quality tracking which may be partly encoded in computer programs;

- Procedural Policy (Rules): which are fully encoded in an executable computer language.

The higher layers of policy are more abstract, are specified in natural language and are meant for human interpretation whereas Process and Procedural policy may be automated so need to be more formally specified.

We have also recognised that the higher levels of policy tend to be more abstract and more applicable to human than automated interpretation, but we have not defined specific policy levels. However the 5 levels of policy shown in Figure 2 correspond roughly to the lower 5 levels described above.

3 RESPONSIBILITY

3.1 Responsibility Relationships

As we have already implied, there is a close relationship between making policies in a policy hierarchy and giving responsibility for a task. Indeed, the two are often synonymous; making a lower-level policy as part of a plan to achieve a higher-level policy is often described as 'giving responsibility' to someone.

However, **responsibility** is a very complex concept; a dictionary definition of 'responsible' includes: being in charge or control, and liable to be called to account; answerable; deserving the blame or credit for; being a free moral agent; morally accountable; able to pay; and respectable-looking! Clearly many of these notions are quite inappropriate for our purpose, and we shall confine ourselves to those aspects of responsibility which appear to be useful for distributed system management. However, we need to emphasise the point even more strongly than in the case of 'policy' that we are not attempting to discuss all aspects of a concept, but are selecting a restricted part of it which will be useful for our purpose.

The subject of a policy can be viewed as the entity responsible for carrying out the goals or actions defined by the policy. Thus the specification of management responsibility can be modelled using our policy objects. Note that this is a different approach from that taken by Strens & Dobson (1992), which advocates the use of responsibility, rather than actions, as the basis for determining requirements for an information technology system. The aspect of responsibility on which they concentrate is 'consequential responsibility', in which one agent is responsible *to* another agent *for* something. Blame or reward is associated with failure or success in the responsibility. Associated with any responsibility is a set of obligations which must be discharged in order to fulfil it. These obligations correspond closely to our imperatival policies, but are not analysed in detail. In this section we concentrate on the 'obligation' aspect of responsibility and analyse some of its characteristics in terms of management action policies.

When a manager is assigned responsibility for an objective there is a need for both a imperatival policy to motivate the manager and one or more authority policies to give the manager the power to perform the action. We discuss the relationship between imperatival and authority policies in more detail in section 4. In the remainder of this section we concentrate on responsibility as a imperatival policy.

We see three different facets of responsibility which may be modelled by three different kinds of imperatival policy:

- Action Responsibility;
- Reporting Responsibility;
- Monitoring Responsibility;

We also note that there is a common English usage of saying that someone is **responsible for** an objective, and **responsible to** someone else for it. This is reflected in the analysis of Kanger (1972) who defined these two aspects of responsibility in deontic logic.

Action Responsibility

Action Responsibility is the simplest of these facets of responsibility. It is intended to convey the sense of being 'responsible for' an objective. We define a subject having **action responsibility** for an objective as the existence of a imperatival policy for this subject and objective. All the policies in figure 3 are expressed in terms of action responsibility.

As we have stated above, we advocate the definition of the policy subject in terms of domains of objects. Primarily this is because it enables policies to be expressed in terms of positions, but it has the added advantage that it also caters for collective responsibility. A imperatival policy with multiple subjects expresses a situation in which all the subjects are obliged to achieve the objective, although as stated before there may be a need for coordination between the subjects.

Reporting Responsibility

In many cases when an action responsibility is assigned to an entity (worker or manager), another manager may be assigned the responsibility for ensuring the objective is achieved and possibly taking corrective action otherwise. This can be achieved by the worker reporting to the manager on completion or at regular intervals. We call this **reporting responsibility**. It is equivalent to 'responsible to' in the English Language.

We can extend figure 3 to include reporting responsibility, if we assume that each subject has both reporting and action responsibility for the objectives. Two policies apply to the System Administrator; he is required both to do the back-up and to report to the Manager of Department D on its result. See figure 4a. However, this is too simple, as he must actually carry out the following actions in sequence:

- Attempt his objective;
- Assess the result;
- If successful, report success to D's Manager;
- If the result is a failure, the policy may simply require the subject to report failure upwards. Alternatively it may require the subject to repeat the attempt a specified number of times before reporting failure.

There need to be a means of linking sequences of actions, with intermediate testing of results, as part of the language of policy definition. This is a task for further research.

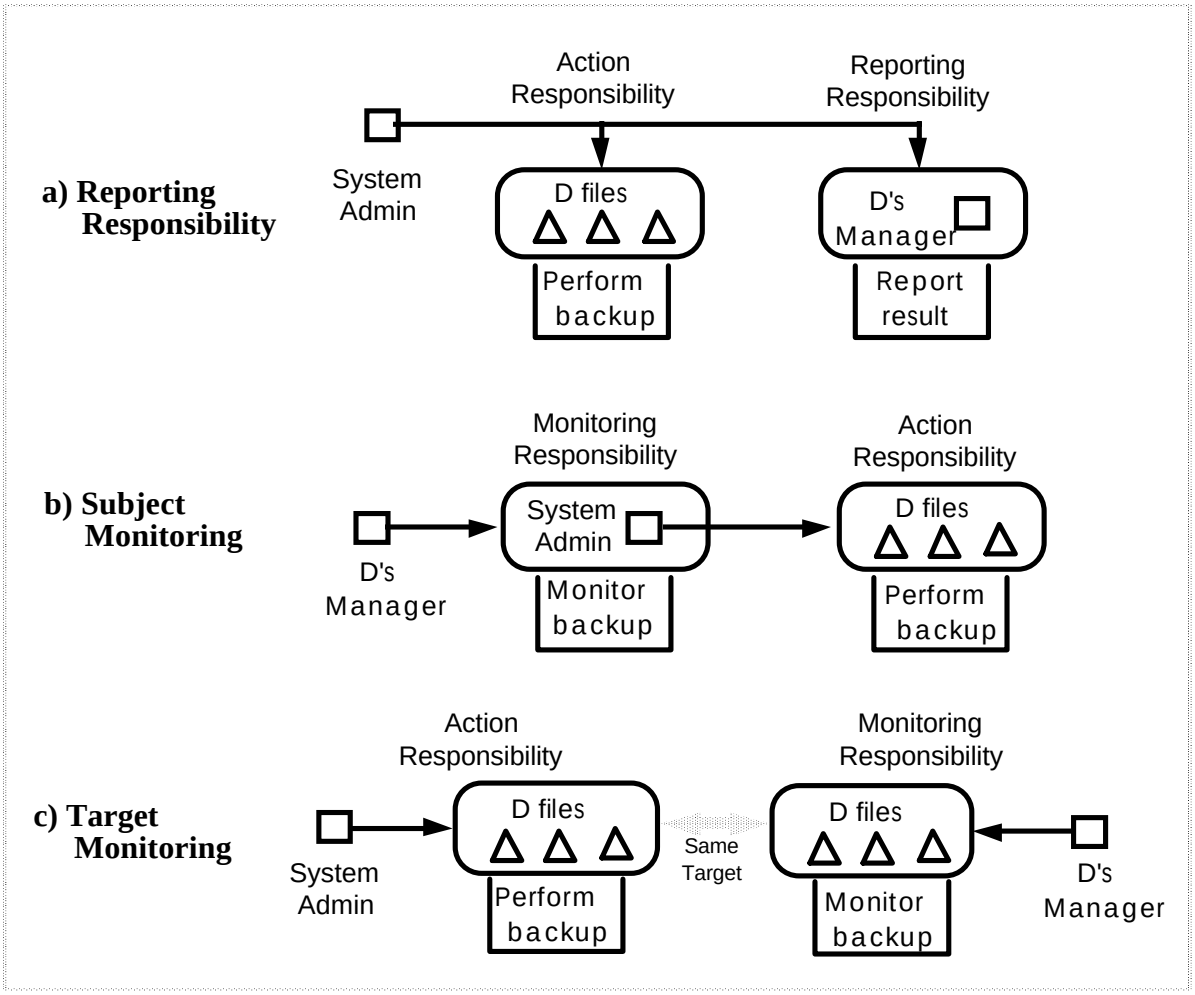

Figure 4 Facets of Responsibility

Monitoring Responsibility

This has the same effect as reporting responsibility, but with D's manager being a subject rather than a target.

i) **Subject monitoring** is linked to an action responsibility, where the action subject is the target of a monitoring policy. In figure 4b, D's Manager is responsible for querying the System Administrator to determine whether the backup policy has been satisfied.

ii) **Target monitoring** occurs when targets of the action and monitoring policy are the same. The success or failure of the action can be deduced from the state of the target objects. In figure 4c, D's manager checks the files to see whether they have been backed up.

3.2 Accepting Responsibility

All our discussion on this subject have emphasised that we are primarily interested in policies which are effective. One aspect of effectiveness is that there is no point in setting up a policy requiring actions to be taken, unless the policy subject in fact accepts the imperative to perform them when required. How can we ensure that a policy subject **accepts responsibility**? In this discussion we have to treat automated and human management differently but compatibly.

Imperatives for Automated Managers

It can be assumed that automated managers are well behaved. As soon as they are made aware of a policy they will begin to perform the actions required to satisfy the policy. There are two ways in which they can be made aware of the policy.

i) The policy creator performs an operation on the manager object to transfer the policy to it. This is the approach taken in (DOMAINS 91).

ii) The policy is stored as an attribute of the subject domain and the manager queries all domains of which it is a member to find out what policies it is responsible for. A manager can be a member of many domains and therefore the subject of many different policies.

To avoid conflict of authority (see section 4.2) the authority for the manager to perform its actions is specified by means of an authority policy with appropriate goals and target objects. The manager object is made a member of the subject domain of the authority policy when it is assigned responsibility.

Imperatives for Human Managers

If human managers are well behaved then simply making them aware of the policy would be sufficient and the above discussion would also apply to humans. However, causing humans to accept responsibility is not so straightforward. It is notorious that policies for human subjects which oblige them to achieve an objective may actually be ineffective in causing them to do so. When a imperatival policy with a human subject is created there has to be a means of ensuring that the motivation to perform it actually exists. This is not a new subject; vast amounts of effort have been spent in attempting to solve the problem. Clearly reward and/or punishment may be used. When the reward or punishment for achieving or failing to meet an objective is formalised, this is described as a contract, and contracts have formed the basis of a number of approaches to Computer-Supported Cooperative Working, e.g. Dowson (1987).

There are a number of different ways in which one human can motivate another. They include: contracts to carry out specific tasks, e.g. a contract with a company to take and store tapes off-site; and contracts by which one person can direct another to perform tasks within defined limits, e.g. employment or subcontract labour. The only necessary relationship between the creator of a policy and the policy subject is that the creator should be able to have some reason to expect that the policy will be carried out, i.e. that the subject will accept the responsibility. Clearly the level of expectation must be suited to the circumstances.

There is also the possibility that an automated manager will create imperatival policies which apply to humans. One can envisage an automated intelligent backup manager changing the frequency of making backups. This would then require alterations to the policy which requires a human to take tapes to storage, to change its frequency. The automated manager would need to inform the human of the changed objectives and obtain the human's acceptance of the new policy.

4 AUTHORITY AND IMPERATIVAL POLICIES

Although we are discussing hierarchies of imperatival policies, authority has an impact upon them in two ways:

- The existence of a situation in which a subject is obliged but not authorised to achieve an objective is an indication of potential conflict;
- There need to be limits on the creation of imperatival policies.

4.1 Conflict of Imperatival and Authority Policies

There is a potential conflict arising from any situation in which an objective is obliged but not authorised. This is a potential, not an actual conflict, because it is often sensible to set up objectives before acquiring the power to achieve them, but it is clearly undesirable for objectives to be impossible to achieve; they should either be achieved or withdrawn.

This kind of conflict is relevant to policy hierarchies because, in parallel with the process of refining objectives and creating lower-level policies, it will be necessary to ensure that the appropriate authority policies are in existence or are created. Authority policies always refer to actions, not to high-level goals, and it is not always possible to identify whether the goals of imperatival policies are authorised until they have been refined to their lowest level.

In practice the approach will depend upon the seriousness with which conflicts of this kind are regarded. In many situations with humans, conflicts between obligation and authority are resolved only when they are encountered, whereas it may be important to prevent them in advance for automated managers, as described above.

4.2 Authorisation of Operations on Imperatival Policies

Since policies are themselves regarded as objects, authority is required in order to create, alter and destroy them. So if a subject refines a high level policy to a set of lower level policies, he will need appropriate authority to create them. It is necessary to control the scope of the subject, the target and the actions of a newly created policy. The ability to create policies and their scopes is itself a matter of policy - not itself a management action policy, but a policy about management action policies (PAMAP). If the computer system were simply a documentation aid for humans, no restrictions would be needed. However, we are interested in policies which are actually used to influence system actions, and therefore the ability to create them must be restricted.

If we consider the effect of creation of a human imperatival policy, this will give some indication of the sort of restrictions which should be applied. In a human business organisation, if A orders B to do something, the following conditions should be met[2]:

- A is B's manager or supervisor, that is the person who B recognises as authorised to direct him to carry out tasks;
- The objective is within the scope of A's objectives;
- B has the necessary power to carry out the objective.

The first condition is needed to ensure that B will be motivated to carry out the task. The second condition is intended to ensure that A only uses his staff for proper tasks. The third condition is intended to prevent the conflict between obligation and authority which was mentioned above, and is not discussed further.

In an automated system one possible approach to ensuring that the first two conditions are met is to define the scope of authority of subjects, whether human or automated, in terms of the subjects who they manage and the objectives which they are required to achieve. We have used this approach in (Moffett & Sloman 1991a) to determine the constraints which should be placed on delegation of authority, and we sketch here a similar approach which could be used for the delegation of responsibility.

A Management Role Object[3] (MRO) defines the type and scope of a manager's role. Policy (PAMAP) decisions can then be made about a general role rather than particular posts. Suppose that there is a role type called Sales Manager. Policy decisions can be made about the general Sales Manager role - the routine tasks he is required to carry out, e.g. produce monthly Sales Reports, and the authority which he has, e.g. to direct his staff and to make purchases. An instance of a MRO whose **type** is Sales Manager is created which defines the North Sales Manager (currently John Smith) to be the **role occupant**, its **subject scope** to be the Northern Office sales staff, and its **target scope** (objectives) to be selling widgets in the North of England. See figure 5.

---

[2] This is of course oversimplifying; people do, or refuse to do, things for many reasons which are outside the formal structure of the organisation.

[3] Referred to as a 'role domain' in our earlier paper, but having the same basic purpose.

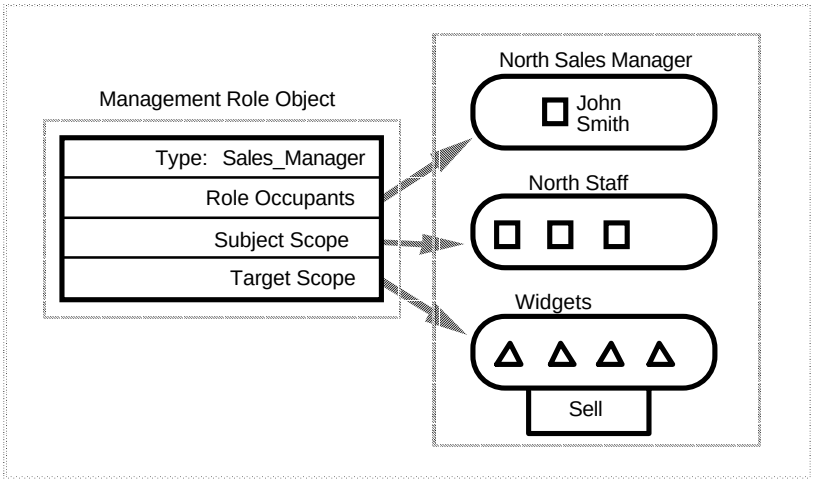

Figure 5 A Management Role Object (MRO)

There is then a general policy (PAMAP), enforced by the system, which authorises Managers of any kind to create imperatival polices only if the first two conditions above are met, i.e. that they can only create them if the policy subject is within their subject scope and the policy objectives are within their target scope. So the only people who John Smith can place obligations on are members of the Northern sales staff, and only for them to perform actions concerned with selling widgets. 'Selling' is presumably a high-level goal which may be refined into the various actions performed by sales staff, from market research to closing contracts with customers.

4.3 Hierarchies of Authority Policies

The primitive modalities - imperatival and authority - are clearly similar to the obligation and permission of deontic logics, but we are continuing to use different terms in order to avoid the possibility that readers might assume that 'ought implies can', as in standard deontic logic. An aspect of the modalities which does, however, disturb us, is their lack of intuitive symmetry. It is perfectly reasonable to set up a policy *obliging* a manager, e.g. to 'safeguard his assets', provided the assets can be identified. The detailed actions to achieve this will be elucidated through the hierarchical process of defining lower-level goals which refine the concept of 'safeguard'. On the other hand, using the present model we cannot meaningfully set up a policy which *authorises* him to 'safeguard his assets'. Authority policies are intended to have immediate applicability to allow or disallow actions, and do not mean anything when applied to higher-level goals. They can only be set up after we have gone through the hierarchical decomposition process, and understand in terms of detailed actions what 'safeguard' means in this case. Possibly there is a need to define a meaning to authority policies which refer to higher-level goals. This would imply extending the discussion of hierarchical policy objectives beyond imperatival policies to authority policies also.

5 AUTOMATED SUPPORT FOR POLICY HIERARCHY ANALYSIS

As we have said above, the ideal is automatic policy analysis and generation, but this is limited by the extent that human intervention is required in the course of refinement. In this section we take a more formal approach to hierarchies of objectives, in order to identify what aspects of refinement can be stated by means of general rules, and what must be provided by human input. The general rules could then be used as the basis for an expert system to provide support for the analysis. We use Prolog syntax, in which the general rules are represented by Prolog predicates over variables, and human input by a database of Prolog facts and predicates over atoms. In order to improve readability, variables are in italics.

The aims of policy hierarchy analysis are to determine whether:

i) The collected lower-level objectives will completely achieve the higher-level objective which they purport to refine;

ii) There is any conflict between the objectives;

iii) There is a imperatival policy, with a subject, for each objective;

iv) There is an authority policy which empowers the subject to achieve the objective.

We are concentrating here on the first of these tasks. The second task has been addressed in (Moffett & Sloman 1993a), and a summary of it is included below. The third and fourth tasks appear to us to be relatively straightforward, once the issues discussed in sections 3.2 and 4.1 have been dealt with.

Our approach to the first task is first to make some preliminary definitions, and then to address each of the main methods of refinement in turn. It is of course necessary, in generating hierarchies, to decide which method of refinement to approach first. At present we have no suggestions about how this can be given automated support, except by making 'trial and error' easier.

Preliminary Definitions

Goals and actions are defined as facts, e.g.:

```
goal(goal1).
action(action1).
```

In addition there is a general predicate which states that actions are (the lowest level of) goals:

```
goal(_action):- action(_action).
```

The target objects of an objective are sets (Prolog lists) of objects, typically defined by unification, e.g.:

```
target_objects1 = [object1, object2, object3].
```

Objectives are defined as facts of the following form:

```
objective(goal1, target_objects1).
```

Our main interest is in an objective being completed. This may be stated as a fact about an action-related objective, e.g.:

```
completed(action1, target_objects1).
```

However, the completion of a higher-level objective is stated in the form of a predicate about lower-level objectives:

```
completed(objective1):-
    completed(objective2), ....
    completed(objective_n).
```

We outlined the ways in which objectives could be refined in section 2.3, and here they are discussed in terms of formal refinement steps.

5.1 Partitioned Targets

Partitioning of Targets is a strategy of dividing up the target objects of an objective while retaining unaltered goals. There is one prerequisite of completeness of an objective in this form of refinement: completeness of target objects. If its goal is satisfied for every individual in the set of target objects, then it is satisfied for the whole:.

```
completed(_goal, _target_objects):-
    objective(_goal, _target_objects),
    forall(on(_object, _target_objects), completed(_goal, _object)).
```

It follows from this rule that, if we divide up the target objects into subsets which are completed, and that the subsets 'cover' them, the objective is completed.

There are two observations to be made about this rule. First, and crucially, its correctness is proved or disproved by reference to experience. Intuitively, it is saying something like 'the whole is equal to the sum of its parts'. What if we could find an objective where 'the whole is more than the sum of its parts'? Then satisfying it for its individual parts would not satisfy it for the whole. So there may be exceptions to this rule.

Second, the rule does not state that the cover has to be an exact one; 'overlaps' are allowed, so that if the target object set of the objective is refined into 'subset' objectives, a single target object may be member of more than one subset. This could have two consequences. First, it may be wasteful - there is no point in having two back-up copies of a file. Second, it could actually violate the objective; if the objective is 'Pay creditors (once)', it should not be done twice for any creditor. If overlaps are not permitted, we need to introduce the requirement for exact partitioning of the target set, which (for brevity) we do not cover here.

An approach to partitioning targets which we advocate is by means of management domains (see section 2.1). It is straightforward to identify complete coverage of a target set, if it is defined in terms of subdomains and it is ensured that every subdomain has been included in the refinement. Identification of overlaps may be achieved by identification of overlapping subdomains, and in some cases there may be an assurance of no overlap because of the constraints which exist on domain operations.

5.2 Goal Refinement

The concept of satisfaction is crucial in the process of goal refinement, moving from a goal to a lower-level set of goals for the same target objects. The question of what is required to satisfy a goal is primarily a matter of fact. Human judgement is required, and this is subject to continual revision, depending upon empirical feedback. The adequacy of a safe store, or the most cost-effective method of delivering a defined quality of service, will vary from place to place and time to time.

We define goal refinement as the choice of a set of lower-level subgoals which will together satisfy a higher-level goal, referring to the same target.

Satisfaction is defined by facts such as:

```
satisfies([subgoal1, subgoal2, subgoal3], goal1).
```

Then an objective is completed if the goal of the objective has a set of subgoals, all of which are completed:

```
completed(_goal, _target_objects):-
    satisfies(_subgoal_set, _goal),
    forall(on(_subgoal, _subgoal_set),
        completed(_subgoal, _target_objects)).
```

There may be several ways of satisfying a goal. One way of satisfying back-up requirements is to copy files to tape and store them:

```
satisfies([copy_to_tape, store], backup).
```

It may not be the only way of completely satisfying the goal of backing up. Perhaps there is the alternative, available in some situations, of copying to a remote site instead:

satisfies([copy_to_remote], backup).

Then a manager (human or automated) can choose between two different methods of backing up, both of which will completely satisfy the objective.

Goal refinement requires a great deal more human input to be provided than for partitioning of targets, but facts about satisfaction can typically be applied to any set of target objects. Analysis of existing policies can search for goals and their corresponding sets of subgoals. Users generating policies can be prompted with proposed goal refinements derived from these facts.

5.3 Arbitrary Refinement

In arbitrary refinement both the goal and the target are quite different from the higher-level objective, but it is known as a fact that satisfaction of this objective will partially satisfy the higher-level objective. For example, we cannot consider System A's file to be protected unless its remote users go through a secure log in procedure:

```
completed(confidential, system_A_files) ->
    completed(secure_log_in, system_A_dial_in_users).
```

Any arbitrary refinement depends upon specific knowledge, though even this may be able to be generalised to some extent. We may wish to make the above objective a general rule:

```
completed(confidential, files(_system)) ->
    completed(secure_log_in, dial_in_users(_system)).
```

A general rule of this kind can then be used to analyse the completeness of a policy hierarchy. This rule states that a confidentiality policy is not complete unless there is a policy of secure log in for dial-in users, and an analysis tool can check for its existence.

5.4 Procedures

An objective may be completed by the completion, in sequence, of the objectives in a **procedure**. We stated above, as a goal refinement, that one way of satisfying back-up requirements is to copy files to tape and store them. This was of course incomplete, because the sequence of these actions is important. This should have been stated as a procedure:

```
completed(backup, _files):-
    completed([copy_to_tape, _files),
    completed([store], _files),
    completed_before(([copy_to_tape, _files), ([store], _files)).
```

The facts about which the order in which objectives are planned to be completed are provided as Prolog facts, e.g.:

```
completed_before((objective1), (objective2)).
```

Sequences of objectives may be defined as facts, or in terms of sequences of goals, e.g.:

```
seq_of(([copy_to_tape, _files), ([store], _files)):- .
```

Completion of an sequence of objectives requires their completion in order.

completed(seq_of([*_objectives*])
forall((on(*_subobjective*, *_objectives*), completed(*_subobjective*)),
forall((on(*_subobj1*, *_objectives*), on(*_subobj2*, *_objectives*),
precedes(*_subobj1*, *_subobj2*, *_objectives*)), [4]
completed_before(*_subobj1*, *_subobj2*)).

So the system requires all the elements of a procedure to be completed in sequence, for it to be regarded as completed.

5.5 Delegation of Responsibility

The discussion in this section has so far concentrated on refinement of objectives but, having arrived at a set of objectives which will complete a higher-level objective, it is necessary also to ensure that subjects have an obligation to achieve them. We have discussed in section 4.2 the limits which may be set upon the delegation of responsibility, by limiting the scope of imperatival policies which a manager can create.

When a manager is proposing to generate policies, the support system can ensure that these are done within the limits of his scope.

5.6 Conflict Analysis

Support tools can analyse whether there are any conflicts between obligation and authority for subjects of existing or proposed policies. A framework for more comprehensive policy conflict analysis has been provided in (Moffett 1993), to which the reader is referred. A summary of it is provided here.

There are several well-known phrases describing policy conflicts. **Conflict of interests** describes a situation where a single person has tasks relating to two different enterprises, and carrying out both together conscientiously may be impossible. **Conflict of duties** is the dual situation; it describes a failure of the control principle of **separation of duties** requiring at least two different people to be involved in carrying out important transactions. **Conflict of priorities** occurs when the resources available are not sufficient to meet the demands upon them. Other forms of conflict are more primitive, and are typically (but not always!) avoided by human managers, for example: an action is simultaneously permitted and prohibited; or, someone has a duty to carry out an action which is prohibited.

Human managers recognise, avoid and resolve conflicts by a combination of formal and intuitive rules and by informal negotiation. They do not always do it very well. Automated systems are forced into a much more formal approach, and there can at worst be a complete failure of the system if conflicts are not dealt with correctly.

The concept of the **overlap** of policies is crucial to the analysis of relationships between them, where overlap between two sets of objects is defined to exist when there is at least one object which is a member of both sets. Management action policy conflicts are not yet well understood, and we do not claim to describe all possible kinds of conflict. However, the policy model has opened up the possibility of systematic analysis of policy conflicts. This analysis is at an early stage, but a first step has been taken by recognising that the overlap of objects – either or both of policy subjects and target objects – between policies is a necessary condition for conflict. If there are no objects at all in common between two policies, there is no possibility of conflict. We classify conflicts as shown in figure 6. The major distinction is between **conflict of modalities** and **conflict of objectives** in imperatival and authority policies.

One form of conflict of modalities is one in which the same subject is simultaneously obliged and inhibited (or permitted and prohibited) for an objective. The other form has been discussed above, in which the subject

---

[4] precedes(Element1, Element2, List) is a predicate which is true if Element1 precedes Element2 on List.

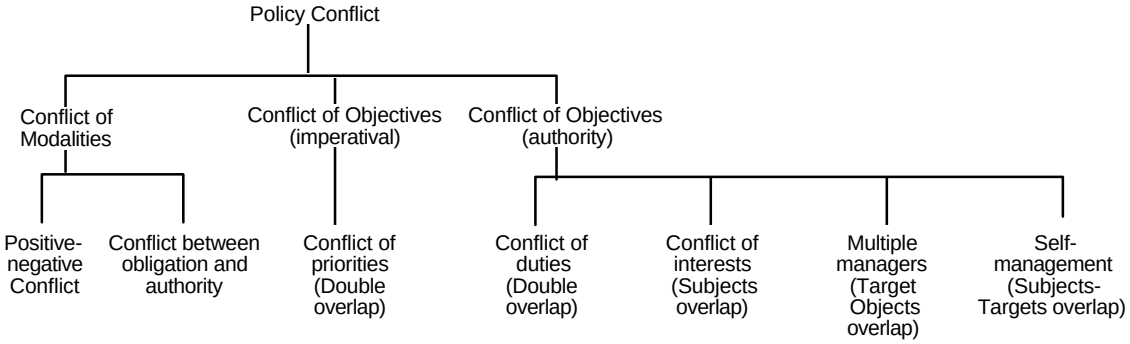

Figure 6 Classification of Policy Conflicts

is obliged, but not authorised, for an objective. Conflict of modalities can be recognised independently of the exact identities of the subjects and objectives of the policies.

We have only identified one form of conflict of objectives in imperatival policies, but this is important for the analysis of policy hierarchies; that is the conflict of priorities, where there are limited resources and satisfaction of one objective makes the satisfaction of another impossible.

Conflicts of objectives in authority policies have several different forms, which emerge from the analysis as corresponding to the ways in which policies may overlap. They are: Conflict of duties (double overlap), Conflict of interests (Subjects overlap), Multiple managers (Target Objects overlap), and Self-management (Subjects-Targets overlap).

6 CONCLUSIONS

Most of the research on management for distributed systems has concentrated on management mechanisms related to Network Management or Operating Systems. However, in order to automate the management of very large distributed systems, it is necessary to be able to represent and manipulate management policy within the system. It must be acknowledged that the whole area of management policy is very complex, and that our model will not be able to cover all aspects of that complexity. The concept of a generic policy object does, however, seem suitable for many aspects of obligation and authority. This paper has attempted to shown how our model clarifies many of the issues relating to refinement of high-level policies into a set of lower-level policies to form a policy hierarchy. We have shown that some aspects of responsibility can also be modelled by our generic policy objects.

The next step forward is practical implementation of these theoretical concepts. Also, several areas requiring further research have been identified, including:

- The avoidance of conflict between multiple policy subjects;
- The need for an enhanced language of policy definition, to enable the expression of concepts such as action sequences, conditional branching and looping;
- Examination of how the proposed rules for refinement of objective work in a practical application;
- Further exploration of the relationship between imperatival and authority policies, in relation to conflicts between them, restrictions on the creation of policy hierarchies, and the authorisation of high-level goals.

The ultimate aim of this work is to develop tools which can aid in the automated management of very large distributed systems. In particular we intend to develop tools for the specification and analysis of policy hierarchies. The use of a uniform standard model for management action policies has enabled us to present an approach to determining whether lower level policies satisfy the higher level ones which they purport to refine and to detecting conflicts between policies. This is just an initial step along the road to automated policy specification and analysis.

ACKNOWLEDGEMENTS

We acknowledge the contribution of colleagues in the High Integrity Systems Engineering group at York University and the Distributed Software Engineering Section at Imperial College in stimulating and criticising the ideas of this paper. We acknowledge financial support from SERC SCHEMA Project (GR/G49531), DTI ESF Project (IED/ 4/410/36/002), Esprit SYSMAN (7026) and IDSM (6311) projects.

REFERENCES

Dobson J.E. & McDermid J.A. (1989), A Framework for Expressing Models of Security Policy, IEEE Symposium on Security & Privacy, May 1989, Oakland, CA, IEEE Computer Society Press, pp 229-240.

DOMAINS (1991), DOMAINS Basic Concepts V 2.0, (Esprit Project 5165), Philips Forschungslabor, Weisshausstrasse, 5100 Aachen, Germany.

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

Kanger S. (1972), Law and Logic, Theoria, vol 38 (1972), pp 105-132.

Masullo M.J., Calo, S.B. (1993), Proc. IEEE Workshop on Systems Management, UCLA, California, April 1993.

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

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

Moffett J.D. & Sloman M.S. (1991b), The Representation of Policies as System Objects, Proceedings of the Conference on Organisational Computer Systems (COCS'91) Atlanta, GA, 5–8 November 1991, in SIGOIS Bulletin vol 12, nos 2 & 3, pp 171-184.

Moffett J.D. & Sloman M.S. (1993a), Policy Conflict Analysis, to appear in Journal of Organizational Computing.

Moffett J.D. & Sloman M.S. (1993b), User and Mechanism Views of Distributed System Management, IEE/IOP/BCS Distributed Systems Engineering, Vol. 1, No. 1, August 1993.

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

Strens R. & Dobson J.(1992), On the Modelling of Responsibility, Computing Laboratory, University of Newcastle upon Tyne, Newcastle NE1 7RU, UK

Yemeni Y., Goldszmidt G. & Yemeni S. (1991), Network Management by Delegation, in I. Krishnan & W. Zimmer (eds), Integrated Network Management II, North Holland (April 1991) pp 95-107.