Journal of Organizational Computing, vol 4 no 1, pp 1-22 (1994)

Policy Conflict Analysis in Distributed System Management

12 April 1993

Jonathan D. Moffett
Department of Computer Science
University of York, UK[1]

Morris S. Sloman
Department of Computing
Imperial College of Science Technology and Medicine
University of London, UK

ABSTRACT

Distributed system management is concerned with the tasks needed to ensure that large distributed systems can function in accordance with the objectives of their users. These objectives are typically set out in the form of policies which are interpreted by the system managers. There are benefits to be gained by providing automated support for human managers, or actually automating routine management tasks. In order to do this, it is desirable to have a model of policies as objects which can be interpreted by the system itself. The model is summarised.

It is clear that there is the potential for conflicts between policies. These conflicts may be resolved informally by human managers, but if an automated system is to recognise them and resolve them appropriately it is necessary first of all to analyse the types of conflict which may occur. We analyse the types of overlap which may occur between policies, and show that this analysis corresponds to several familiar types of policy conflict. Some possible approaches to the prevention and resolution of conflicts are suggested, and this work is put into the context of other work on policies and related areas, including deontic logic.

Keywords

Management policy, policy conflicts, authority, conflict resolution, distributed system management.

[1] Address for correspondence:
Department of Computer Science, University of York, Heslington, York YO1 5DD, UK
Email: jdm@minster.york.ac.uk

1. INTRODUCTION

This paper describes a model of policies for use in distributed system management, and analyses some possible forms of policy conflict in terms of the components of the model.

1.1. Distributed System Management

Large distributed processing systems are becoming increasingly important for organisations to manage their own activities and interact with others. They typically consist of multiple interconnected networks and span the computer systems belonging to a number of different organisations.

Active management, rather than fire-fighting when problems occur, is essential for systems of this kind for several reasons. First, many organisations now depend upon distributed systems in order to function, and they have developed from supportive to operational roles, so there has to be a positive approach to ensuring that they function adequately. Second, in many cases the system is not controlled from one central point, but has to be maintained by the cooperative effort of several independent managers. Third, distributed systems are highly complex in several dimensions: they have diverse components supplied by diverse manufacturers; they may be distributed across wide geographical areas, across international boundaries, across different regulatory authorities and across time-zones; they may contain hundreds of thousands of resources and be used by thousands of users.

Distributed system management can be defined briefly as the task of maintaining the service required of a system. In order to make this possible for heterogeneous systems, and to cope with complexity, a set of standards for Open Systems Interconnection (OSI) management [1] have been developed. They partition the overall management into areas of functional responsibility. The main functional areas defined by OSI are: Configuration Management to control installation of both the hardware and software components within a distributed system or application; Performance Management, concerned with optimisation of performance to improve the service provided to users in terms of better throughput, response times or reliability, or to reduce operating costs; Fault Management, including detection, location and recovery from faults; Security Management to maintain the security mechanisms of the system, e.g. access control, encryption facilities and physical security; and Accounting, which records information on usage of resources and enables suppliers of services to charge for the use of those services. In addition, Monitoring of state, errors, performance and usage information is needed in order to support all the above management functions, although it is not a standard OSI Systems Management function.

The approach to distributed systems management, like that for any other management task, is to take action as far as possible on the basis of general policies, not of particular cases. This implies the generation of policies which apply to abstractly defined situations, and to groups of components and users of the system rather than individual units. For example the same policy may apply to all people in a department or to the set of files pertaining to an application. The grouping of system objects into **management domains**, to which policies may refer, has been described in [2].

1.2. Paper Outline

The paper covers its subject matter as follows. Section 2 introduces the subject of policies and the need to provide a structured model of policies in order to analyse conflicts. Section 3 describes our model of management action policies and their attributes. The fundamental distinction between imperatival policies, which cause actions to be initiated, and authority policies, which give them the power to happen, is made. Policies may themselves be objects which are part of the system, and the operations which can be performed upon them are introduced. The concept of the **overlap** of policies, which is crucial to the analysis of relationships between them, is defined. Section 4 describes and analyses several kinds of **policy conflict** in terms of the possible types of policy overlap. In section 5 some possible approaches to the **resolution of conflicts** are introduced. Related work is described in section 6. Finally, in section 7, conclusions are reached about the progress which has been made and the research tasks which remain.

2. POLICIES

2.1. Management Policies

All formal organisations have policies, which are defined in the dictionary as 'the plans of an organisation to meet its goals'. They are the driving force behind management. 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 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 authority to) the resources which are needed to carry out the goals. If they allocate money they are typically called Budgets.

A common theme in distributed system management is the need for independent managers to be able to negotiate, establish, query and enforce policies 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 authority. 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 to his users 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 obligation to do so, while the IC Network Manager has the obligation to obtain the information but no authority to do so. The initial situation is shown in figure 1a.

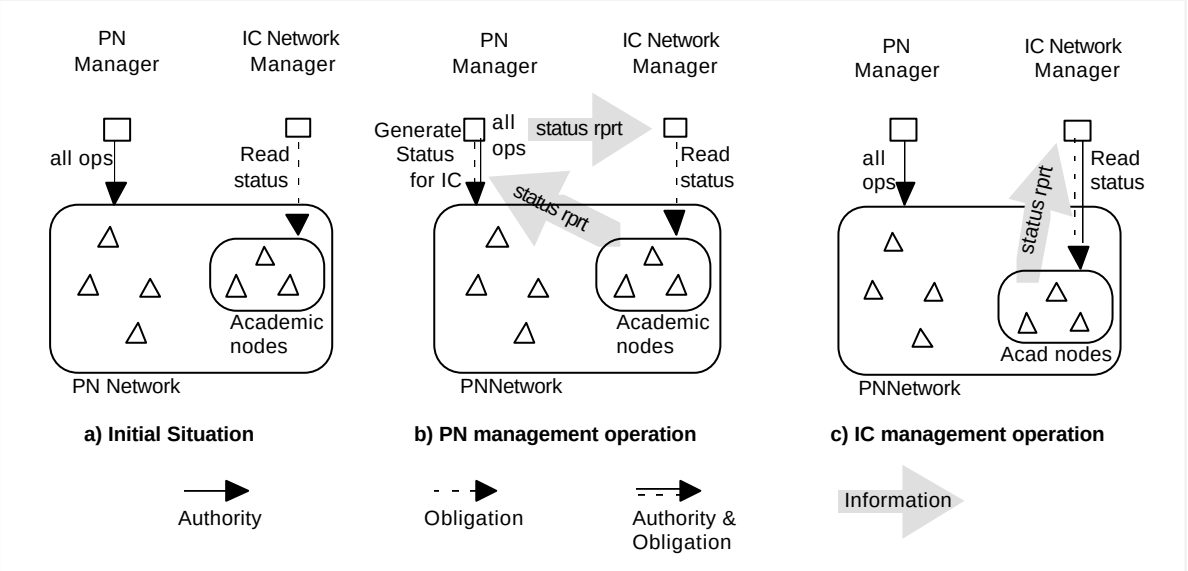

Figure 1 Policies for PN and IC Managers

An additional policy has to be established (created) by the PN Manager to meet IC's requirements. One approach is to create an imperatival policy which causes 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 is to create a policy which gives 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 cause activities to be initiated 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 causing him to do so.

2.2. The Need to Model Policies

We view the activity of system management as implementing the policies of the organisation(s) which run the system. 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. Thus it must be possible 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. An important factor in distributed system management is the potential benefit to be gained from computer-assisted support for it or even, in appropriate cases, its complete automation.

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.

'Policy' is a very wide term, and there can be little hope of capturing all kinds of policy in a model. We distinguish here between two particular kinds of policy, while recognising that there may be many others which are of neither kind. **Management action policies** are the main kind which are of interest to distributed system management; briefly, they describe a persistent, positive or negative, imperative or authority for a set of policy **subjects** to achieve **goals** or **actions** on a set of **target objects**. Human subjects or targets can be represented by software objects within the computer system.

However, other policies cannot easily be fitted into this framework. For example, the policy that 'the same person must not be authorised both to enter a payment and sign the payment cheque' cannot easily be modelled as a management action policy. It is most easily described as a **policy about management action policies (**PAMAP policy), because two management action policies are in question: one authorising X to input accounts for payment, and the other authorising X to approve accounts for payment. The PAMAP policy is that the two management action policies must not coexist. This example is considered further in section 4.5.

We do not at present know how to model PAMAP policies usefully, and so where we need to describe one we have to step 'outside the model'. This is not important at present because there is still a great deal of work still to be done exploring ordinary management action policies and their relationships. However, in the long term this is a significant limitation which will need to be overcome.

2.3. Policy Conflicts

A dictionary definition of **conflict** is 'opposition, difference, disagreement', and we will use this as a general guide. 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 authorised and forbidden; or, someone has a duty to carry out an action which is forbidden.

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. This paper is an exploration of how far it is possible to use our model of policies to analyse conflicts with a view to their prevention, identification and resolution.

3. A MANAGEMENT POLICY MODEL

3.1. Characteristics of Management Action 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. However, policies are not concerned with instant decisions to perform an action, instantly carried out. If a manager specifies that something is to be done once only, and instantly, e.g. an order 'Shut the door!', he does not create a policy, but simply causes the action to be carried out. Our definition of policy requires it to have persistence, whether because it defines a single future action or repeated actions, or because it relates to the continuing maintenance of a condition.

3.1.1. Policy Modalities - Imperatival and Authority

As shown in the example above, we distinguish between policies which are intended as imperatives to initiate actions, and policies which give or withhold authority for actions to take place. **Actions** are operations which are performed by agents on target objects provided two preconditions are satisfied: imperative and authority:

- **Imperatival** policies are those which cause actions to be initiated (or deterred). A common form of imperative is an obligation which is undertaken by the agent, and many of our examples refer to obligations as a form of imperative;
- **Authority** policies are those which cause actions to be given authority to be carried out.

Figure 2 illustrates our view of the world. Agents are objects which interpret and apply imperatival policies of which they are the subjects. Whenever the conditions of these policies apply, the agent initiates an action, directed to a target object . However, the action will only operate if it is authorised; a reference monitor intercepts all initiated actions and only allows them to proceed (authorises them) if the applicable authority policies permit this. The authorised action operates upon the target object.

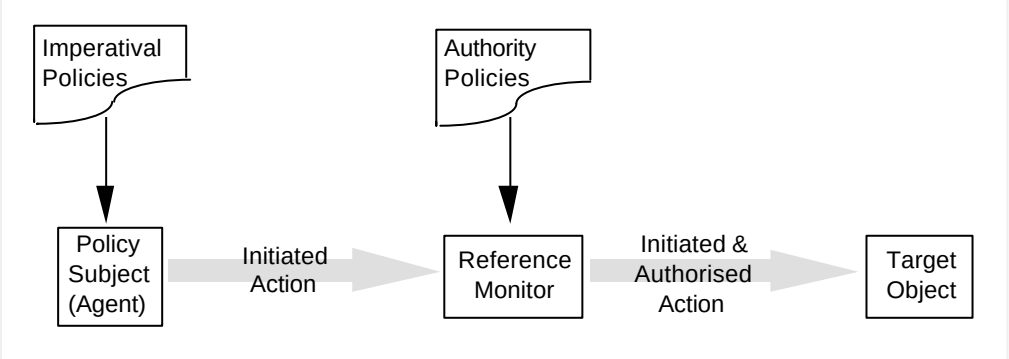

Figure 2 The Roles of Imperatival and Authority Policies

3.1.2. Policy Attributes

Policies, whether concerned with imperatives or with authority, have at least the following attributes: modality; policy subjects; policy target objects; policy goals; and policy constraints. We use a standard graphical convention for illustrating policies (omitting constraints), as shown in figure 3. It will be seen that, for graphical convenience, the subjects and target objects are shown in standard Venn diagram convention, while the set of goals is shown as a list attached to the policy modality. We use the convention that subjects are represented by squares and target objects by triangles.

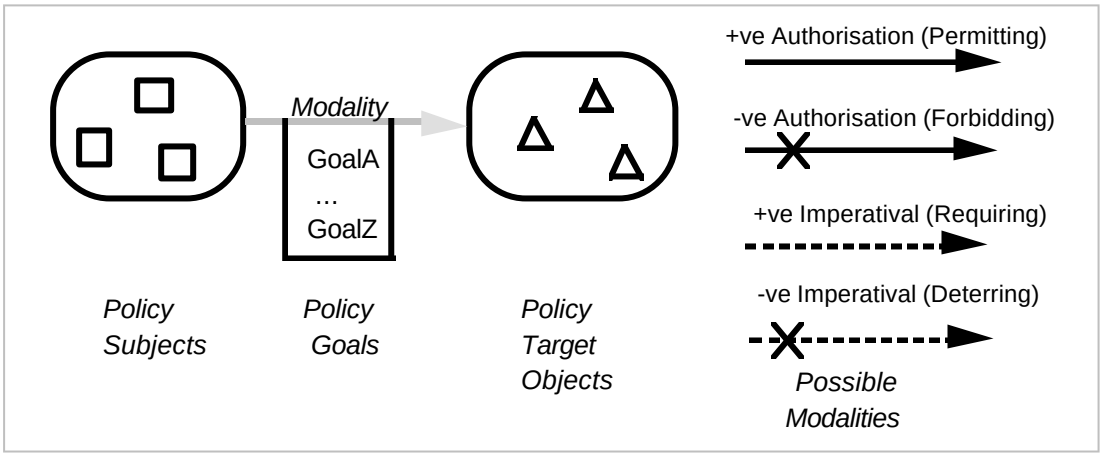

Figure 3 A Management Action Policy

Modality

A policy has one of the following modalities: **positive authority** (permitting), **negative authority** (forbidding), **positive imperatival** (requiring or obliging), and **negative imperatival** (deterring). We do not exclude the possibility of other useful policy modalities being proposed, but these are adequate for the present analysis. As mentioned above, we regard an obligation as a special form of imperative.

Policy Subjects and Target Objects

Policies are about organisational goals, which need someone to achieve them. All policies in this model have a **policy subjects** attribute which defines a set of users. The policy subjects are the people to whom the policy is directed, i.e. who have the imperative or authority to carry out the policy goal within the limits defined by the policy constraints. Where a policy has been automated as a computer system command we regard the user who will input the system command as the policy subject. Note that many policies are by implication directed to all users, possibly constrained by some predicate, and the value of the policy subjects attribute is then the set of all users.

The **policy target objects** attribute defines the set of objects at which the policy is directed.

The sets of policy subjects and target objects may each be specified either by enumeration or by means of a predicate which is to be satisfied. Individual policy subjects and target objects are not normally specified, 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 [2].

It should be noted that, although the subjects of both imperatival and authority policies are defined as sets, the typical set membership is likely to be different. The set of subjects of an authority policy simply describes who has the power to perform actions, and no problems arise if there is a large number of members of the set. On the other hand, the set of subjects of a imperatival policy will typically consist of one member only, because usually actions are performed by one person, and when the same goal is allocated to more than one, conflict may result. It may be appropriate for the subject set to be defined as a position, e.g. Security Administrator, with more than one member, but it will then be necessary for members of the position to coordinate between themselves to avoid conflict. This is discussed further below in section 4.6.

Policy Goals

The policy goals attributes can be expressed as a **high-level goal** which specifies what the manager should achieve in abstract terms which do not identify how to achieve the goals. Alternatively the goals can be refined to a set of more concrete **actions** which specify how to achieve the required goal. Actions are specified in terms

of an alphabet of operations which can be performed on objects in the system, and so are amenable to automated interpretation.

An example of a high level goal is 'Department Ds' managers must protect department D's files from loss due to fire or media failure'. This can be refined into the following set of actions:

i) The system is to run a backup program to archive files on Department D's file server to cartridge tape every night at 22.00;

ii) The duty operator is to take the cartridge tape to off-site safe storage every morning at 08.00.

A high-level goal can be refined into many alternative sets of actions. The process of refining a goal to a set of actions is similar to refining a set of requirements into the detailed design of a computer program.

Note that a high-level authorisation policy such as the 'computer manager has authority to order computer equipment' will be refined into a set of lower level policies which authorise the manager to perform specific operations on system objects, e.g.:

read budget;

read suppliers;

new order_transaction;

query payments_made.

There is no inherent ordering relationship between these permitted operations.

Policy Constraints

The Policy Constraints attribute of a policy object places constraints on its applicability. They are predicates which may be expressed in terms of general system properties, such as extent or duration, or some other condition. An example of constraints in 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, for example 'Members of Payroll are permitted to Read Payroll Master files, from terminals in the Payroll office, between 9 am and 5 pm, Monday to Friday'.

3.2. Representing Management Action Policies as Objects

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

- Create a policy;
- Destroy a policy;
- Query a policy.

Authority may be required to perform operations on policy objects. If the computer system is simply a documentation aid, no restrictions may be needed. On the other hand, if the policies are actually used to influence system actions, as in the case of access control policies, restrictions on operations are required. They are discussed in detail for authority policies in [3].

The advantage of representing policies as explicit objects which are interpreted by managers is that it is easier to determine what policies exist and to change them. If necessary, policies can be made read-only to prevent change. However, many systems define policy implicitly by coding it into the implementation of the system or the manager components. Even if it is necessary to encode a policy into an implementation as the only practical means to implement certain policies, there should still be a (high level) policy object to explicitly specify the policy so that it does not get changed with a new release of the system without realising there has been a change of policy.

3.3. Overlapping Policies

Our model of management action policies represents both subjects and target objects as sets of objects. The overlap relationship between sets of objects exists when their intersection is non-empty, as shown in figure 4.

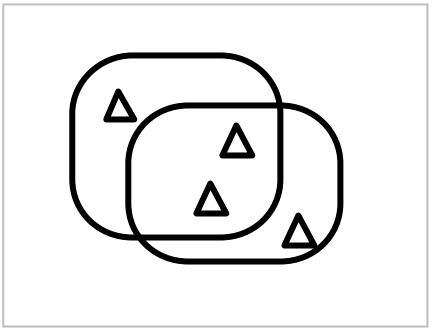

Figure 4 Overlapping Sets

Overlap is crucial to our discussion of policy conflicts as it is our contention that without some kind of overlap between the objects in two policies there can be no conflict between them. We analyse conflicts below using the kind of overlap as our first level of classification. There are several possibilities for overlap between policies, corresponding to various combinations of overlap between objects in the subject and target object sets of the policies:

- **Double overlap** - both the subjects and the target objects of the two policies overlap;
- **Subjects overlap**;
- **Target objects overlap**;
- **Subjects - Targets overlap** - the subjects of one policy and the target objects of another policy overlap.

Overlap of some kind is of course the prerequisite for many kinds of relationship between polices which do not involve conflict. Here are some examples in which the target objects attributes of policies overlap but there is no conflict:

- **Authority hierarchies**. Many organisations have a well-defined authority hierarchy. When a high level manager delegates authority to subordinate managers, this is usually a partitioning of the target objects into subsets assigned to different managers. Obviously the target object set, in the policy which gives authority to the higher level manager, overlaps with the target object sets which have been delegated to the subordinate managers. Conflict is avoided by an imperatival policy deterring the higher level manager from exercising control over the delegated targets, except possibly in case of failure of a subordinate manager.
- **Imperatival policy hierarchies**. When a high level imperatival policy is refined into more concrete lower level policies or sets of actions (as described in section 3.1), there is obviously overlap between the target objects of higher and lower level policies. This does not lead to conflict as only the more concrete lower level policies will actually be translated into action by managers.
- **Responsibility**. As pointed out by Kanger [4] there is a distinction between responsibility *for* and responsibility *to*, when discussing a goal to achieved. We think that it is likely to be useful to analyse the concept of responsibility into two separate imperatival policies which both apply to the same set of target objects. The two policies would have as their subjects, respectively, the manager who is responsible for achieving the goal, and the manager to whom the first manager is responsible. This is the subject of current research.

4. MANAGEMENT ACTION POLICY CONFLICTS

Management action policy conflicts are not yet well understood, and we do not claim in this paper 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 5. The major distinction is between **conflict of modalities** and **conflict of goals**. Conflicts of modalities can be recognised without reference to the meaning of the policy goal, whereas conflicts of goals depend upon the semantics of the goal, or are application-dependent.

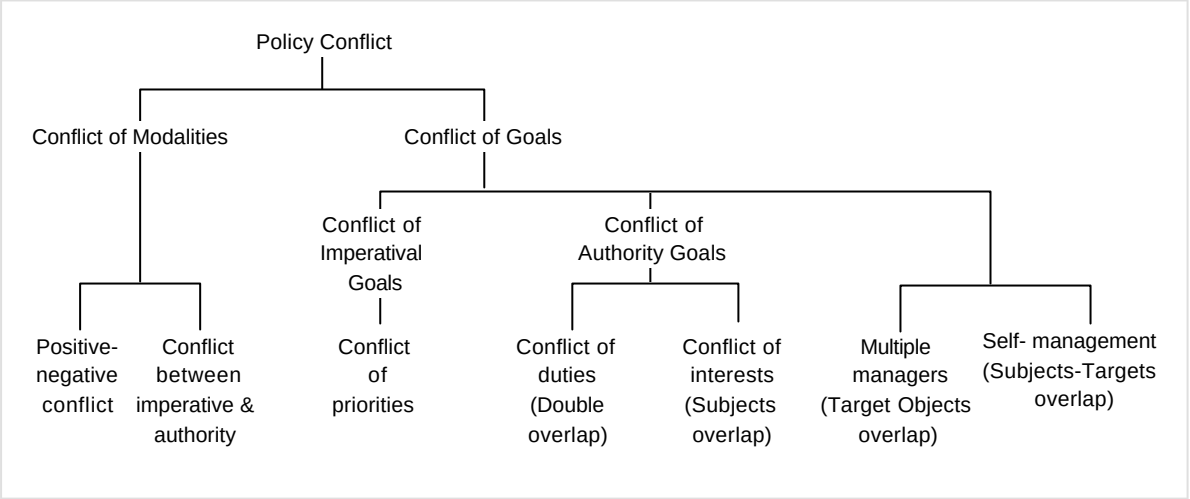

Figure 5 Classification of Policy Conflicts

4.1. Positive-Negative Conflict of Modalities

Direct positive-negative conflict is shown in figure 6. It requires a triple overlap of target objects, subjects and goals. It occurs when a subject is both authorised and forbidden for the same goal on an object, or both required and deterred: for example 'The Accounts Supervisor is permitted to sign payment cheques' *and also* 'The Accounts Supervisor is forbidden to sign payment cheques'. This conflict is very serious, as there is no means of deciding whether the action is to be permitted (or initiated, depending on the modality); unless there is some means of resolution this could lead to a deadlocked situation if the conflict were encountered in an automated system. Conflicts of this kind in access control (authority) policies are discussed in [5].

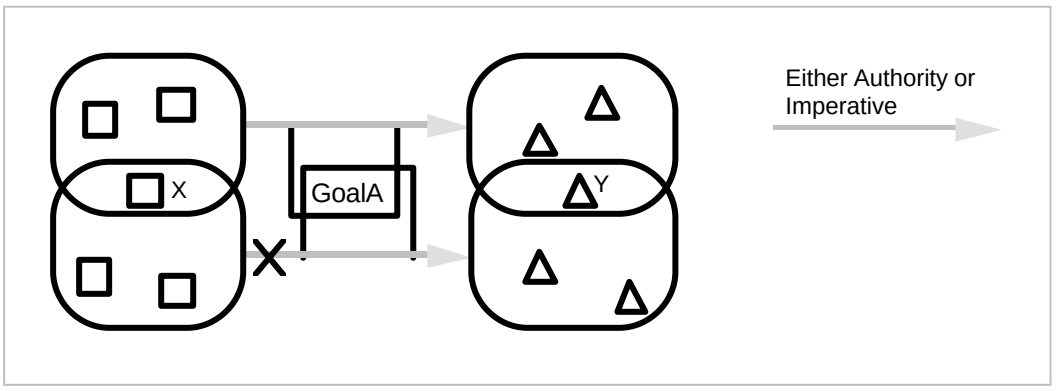

Figure 6 Direct Positive-Negative Conflict of Policy Modalities

Note that this conflict can be detected without understanding the semantics of the particular goal; we can immediately recognise that the following two statements are in conflict:

a) *X* is obliged (/authorised) to achieve *GoalA* on *Y*.

b) *X* is obliged not (/forbidden) to achieve *GoalA* on *Y*.

4.2. Conflict between Imperatival and Authority Policies

Conflict between imperatival and authority policies is shown in figure 7. It also requires a triple overlap of target objects, subjects and goals. It occurs when a subject is both required to initiate and forbidden to carry out an action on an object: for example 'The Accounts Supervisor is obliged to sign all payment cheques which have been approved by the Accounting Manager' *and also* 'The Accounts Supervisor is forbidden to sign payment cheques'. This conflict is less serious because the question of whether the action is to be carried out is settled unambiguously; although it is initiated as a result of the imperatival policy, it will be prevented when the reference monitor intercepts it, because of the negative authority policy (see figure 2). However, it is at least strange if someone is simultaneously obliged and forbidden to do an action. At best this is a temporary situation, e.g. reflecting a partially completed organisational change. At worst the work of the organisation may be brought to a halt.

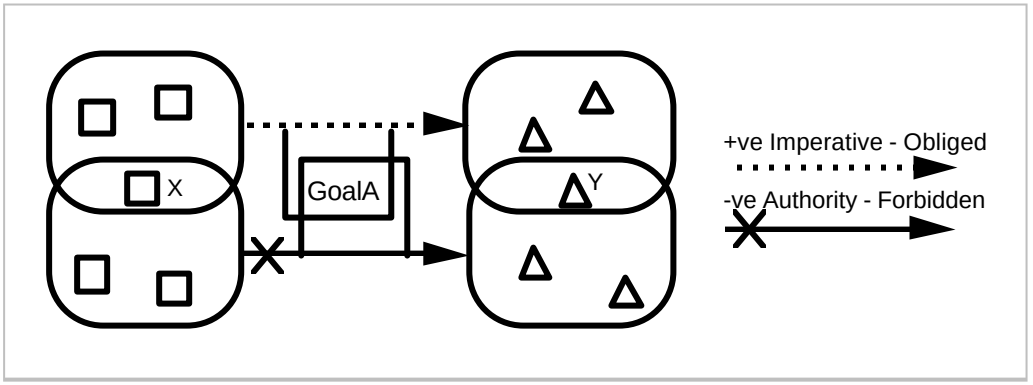

Figure 7 Conflict between Imperatival and Negative Authority Policies

This conflict too can be detected without any application knowledge at all; we can immediately recognise that the following two statements are in conflict:

a) *X* is obliged to carry out *ActionA* on *Y*.

b) *X* is forbidden to carry out *ActionA* on *Y*.

There is of course also the complementary situation, in which a subject is authorised to achieve a goal on an object, but deterred from doing it. We do not regard this as a conflict at all, because it is commonplace to give a manager powers but instruct him not to use them until some future situation arises. However, it is not in general a good thing for subjects to have unnecessary powers, and it is therefore good management practice to detect this situation as part of the management monitoring activity.

4.3. Conflict of Priorities for Resources

The situation of conflicting requirements for a limited or single resource is familiar wherever concurrent access may be attempted, e.g. in database systems. This is a conflict (or contention) between actions, not policies. There is also a familiar situation of competition within an organisation for resource budgets, which may be viewed as a particular kind of authority policy. However, we are not attempting to model contention or competition in this paper.

There is one area of conflict for resources where the policy model may be able to help, although our analysis is at an early stage. **Commodity resources** are resources which have a quantity associated with them, rather than being atomic, so that they can be partially used. Examples are money, time, and memory or disc space. Typical operations that can be performed upon them are **use** and **replenish**, parameterised by a quantity. So if a particular money resource has a value of £400 and the operation *use(300)* is performed on it, its value afterwards will be £100.

Casual reading of a number of organisations' policy statements suggests that many of them are intended to initiate the spending of money or use of other resources. Clearly when two or more policies between them require the use of more resource than is available, there is a **conflict for resources**. This is shown by two

imperatival policies whose goals are each *use* and whose target object is the same money resource. This is shown in figure 8. Further progress on this area of conflict requires a better understanding of how to represent commodity resources in computer systems. Varley [6] provides some discussion of them.

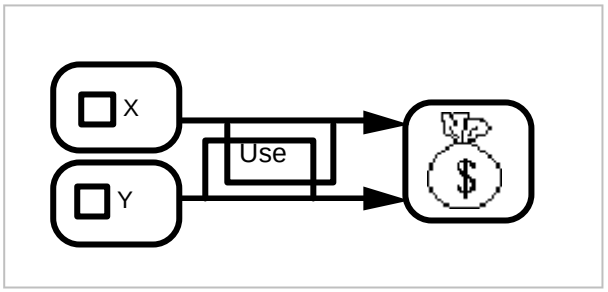

Figure 8 Conflict for Resources

4.4. Conflict of Duties

When two positive authority policies are in the double overlap relationship, in which both the subjects and the target objects of the two policies overlap, there is the possibility that a subject can perform two operations, which are defined by the application as conflicting, upon an object. This is described as a **conflict of duties**, more familiarly known as a failure of the control principle of **separation of duties** described by Clark & Wilson [7]. See figure 9a for the general case; figure 9b illustrates a special case, perhaps more common, in which the two conflicting operations occur in a single authority policy. An example of the principle is 'The same person must not be allowed both to enter a payment and sign the payment cheque' . This is a PAMAP policy, which would be violated by two such policies as 'The Accounts Supervisor is authorised to enter payment information' and 'The Accounts Supervisor is authorised to sign payment cheques', unless further restrictions are applied.

Application knowledge is required to tell whether two goals can give rise to a conflict of duties. If conflicts of this kind are to be recognised in a system, it is necessary to set up a table of pairs of goals which are declared to be in conflict.

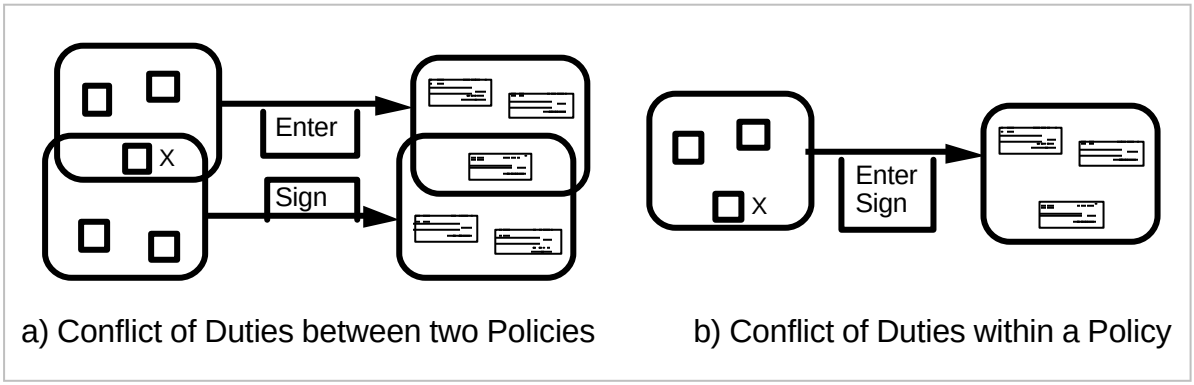

Figure 9 Potential Conflict of Duties

A conflict of duties arises if both subjects and targets of two policies overlap (treating the two goals of figure 9b as two separate policies). The absence of subject overlap is a sufficient condition to enforce separation of duties, and Clark and Wilson discuss this in more detail. Brewer, Nash & Poland [8, 9] discuss how the condition may safely be relaxed further; they would characterise the absence of subject overlap as a **static** separation of duties, and they describe methods of **dynamic** separation of duties which are more flexible in operation.

4.5. Conflict of Interests

When the subjects of two authority policies overlap, this implies that the same subject can perform management tasks on two different sets of targets. See figure 10. In some application-defined circumstances there is a conflict, known as a **conflict of interests**. The best example of this is when a merchant bank is acting as adviser to two different organisations, e.g. on a takeover bid for one client while advising other clients on investment decisions which would be influenced by knowledge of the takeover. There is a public policy (PAMAP) forbidding the use of this knowledge, and declaring the situation to be one of conflict.

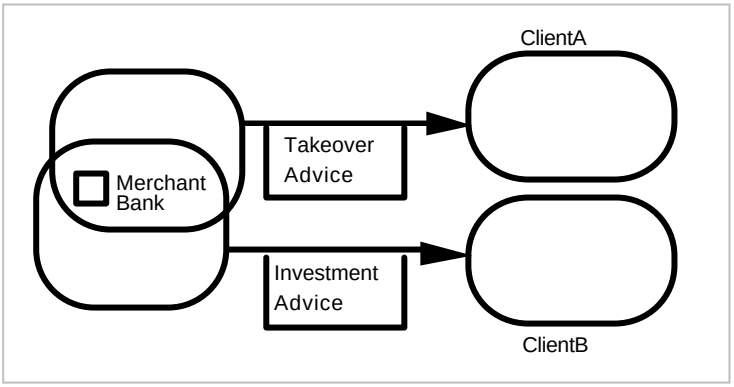

Figure 10 Potential Conflict of Interests

4.6. Multiple Managers

When the target objects of two policies, of either mode, overlap, there is a potential conflict arising from **multiple managers** of a single object, when the goals of the policies are semantically incompatible. For example, if the 'Maintain' operation on a computer entails taking it out of service, and the 'Schedule' operation requires it to be in service, any two policies which oblige subjects to do both simultaneously are in direct conflict. See figure 11.

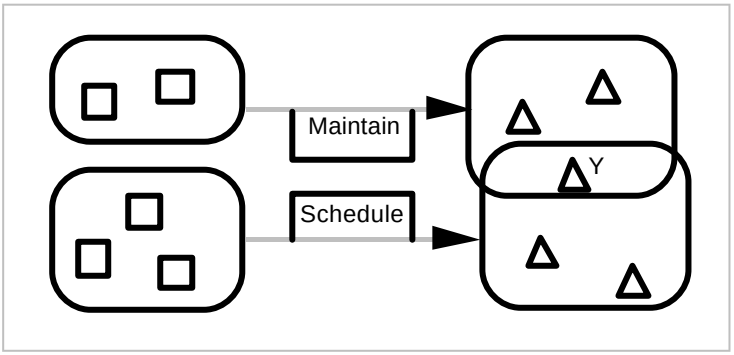

Figure 11 Multiple Managers

There is also a potential conflict arising from multiple managers having authority over the same object, but this is often tolerated. In some cases multiple managers of an object are forbidden on the grounds of potential conflict, e.g. generally each worker has one line manager. In other cases it is positively encouraged, e.g. there are normally at least two people with security administrator authority for a computer system, to cover sickness and holiday absences.

The conflicts which arise from multiple managers being authorised to operate upon a single target object are often controlled by ensuring that there is no simultaneous conflict of obligations. The way in which Security Administrators cooperate is by coordinating their obligations; the backup Security Administrator normally has no obligation to use his authority, and will refer requests for action to the primary Security Administrator. However, when the primary is unavailable, the backup's imperatival policy becomes activated and gives him the obligation to take action. When this sort of coordination is carried out between humans, it may be informal and

even unformulated, but when the managers are automated it is necessary to formalise the way in which the obligations are controlled, by ensuring that the relevant imperatival policy only applies to one subject at a time.

Another source of multiple management is mentioned in [2]. If the same real-world object, e.g. a computer, is represented by two different software objects, e.g. in scheduling and maintenance applications, then there can be interference between the two applications which cannot be dealt with by the system. For example, if the maintenance system has caused the computer to go into a disabled state, there is no possibility for it to be scheduled. This kind of conflict may be serious, because it is undetectable and unresolvable within the computer system. It can only be dealt with at the design stage, by ensuring that multiple representations of real-world objects are avoided.

4.7. Self-Management

The final situation to be mentioned is when the subjects of one policy overlap with the target objects of another policy. We do not know of any interesting conflicts which arise from this kind of overlap when the two policies are different, but when this applies within a single authority policy there occurs the possibility of **self-management**. The situation is of a manager managing himself. See figure 12. This is again a potential conflict which is application-dependent; it may be acceptable for an automated manager to configure itself, but not for a human manager to sign his own expenses.

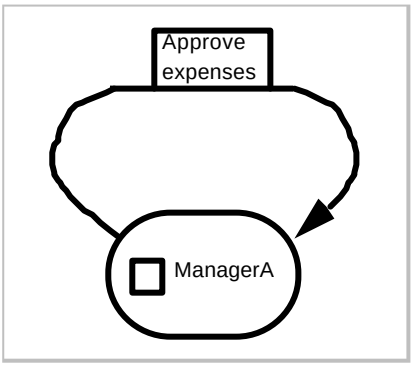

Figure 12 Self-Management

5. RESOLUTION OF CONFLICTS

As observed above, the degree of importance to the resolution of conflicts varies widely. At one end of the scale, it is essential that direct positive-negative modality conflicts should be prevented or that a method of immediate resolution is available. At the other end, many conflicts of duties are tolerated indefinitely.

There are several different levels at which conflicts can be prevented or resolved:

- At the highest level, the language used for system description may be designed to prevent an unresolved conflict arising at all, e.g. our approach to prevention of authority conflicts in this section.
- Conflicts may be detected off-line by a compiler or some other pre-processor. Type checkers (now) and automatic proof systems (when they exist) detect inconsistencies in specifications and programs. Potential conflicts of interest may be detected by application-specific tools which are aware of conflicting goals. The (human) developer can then resolve these conflicts off-line.
- Potential conflicts may be detected on-line in advance, and prevented. For instance, one form of conflict of duties could be prevented by restricting the domain of subjects of an authority policy to a single member at any moment.
- Actual conflicting actions may be detected at the time that they occur. Another way of dealing with conflict of duties could be by detecting that an action is being performed which reverses an action carried out by another subject. It would be an application-specific decision whether to prevent the action or simply to raise a warning to allow the (human) user to resolve the problem.

We discuss two specific approaches to conflict resolution.

5.1. Resolving Positive-Negative Authority Conflicts

Direct positive-negative conflicts must be prevented or resolved. For example, authority conflicts are prevented in [10] by means of a two-level priority scheme. The default, low priority, authority for any action is negative, so that in the absence of explicit positive authority no action is permitted. Explicit authority policies always express positive authority, with high priority. They cannot conflict with each other, as there is no conflict involved in authorising an action twice over. They always have precedence over the default negative authority, so the 'conflict' between the default and explicit policies is always resolved.

Mandatory and Discretionary Security, are two forms of security defined by the Department of Defense (USA) [11]. Mandatory Security is always expressed as a negative authority policy, and Discretionary Security may be expressed by a positive authority policy. When they conflict, there is a PAMAP policy (see section 2.2) that the negative authority of the Mandatory Security policy has higher priority and overrides the positive authority of the Discretionary Security policy. Implicitly, a three-level priority scheme is used to ensure conflict-free policies in this situation: low priority negative default policy; medium priority positive discretionary authority policies; and high priority negative mandatory authority policies.

5.2. Resolving Authority Conflicts by Imperatival Policies

As discussed in section 4.6, on multiple managers, an authority conflict can in some circumstances be dealt with by ensuring that the associated imperatival policies are coordinated. Potentially conflicting actions between two authorisation policies may be prevented by ensuring that the respective subjects are not simultaneously obliged to perform them . For example, the maintenance manager may be authorised to disable a computer and the scheduler to schedule it. If the first occurs it will be impossible for the second to be carried out. The conflict can be resolved by ensuring that imperatival policies for the maintenance manager and the scheduler are not in force simultaneously.

It depends entirely upon the application whether this method of conflict resolution is satisfactory. In the case of multiple managers, e.g. two people acting in the security administrator role, it is a commonly adopted technique of resolution. In other cases it is impossible to be sure that a subject is deterred. In relation to conflict of interests and human agents, it is well known that negative imperatival policies have frequently failed to prevent actions in breach of them; policies which remove the authority, not simply the imperative, may be necessary.

6. DISCUSSION AND RELATED WORK

6.1. Human or Automated Managers?

In the above discussions we have not distinguished between human and automated managers. We assume that automated managers are 'well behaved' and always perform the goals specified by imperatival policies, whereas human managers always have freedom of choice and may refuse to carry out an obligation for whatever reason. We have not attempted to model this freedom of choice and instead assume 'good behaviour' of both automated and human managers.

We assume humans are represented within the computer system by a persistent 'user representation' object for which policies can be specified. When a person logs into the system, an active object (software process) is created to interface to the person's workstation, and this object 'inherits' all policies specified for the user representation object. Initiation of an action can be represented by the human typing in a command, which will only be performed if the human subject has authority for it.

We have not yet attempted to make any formal analysis of the implications of the difference between human and automated managers, nor are aware of any publications doing so. However, a more detailed analysis than this will need to explore the distinction.

6.2. Sociological Approaches

The heart of our model is the categorisation of policies into imperatival and authority policies. We have not seen this distinction made elsewhere in relation to the explicit discussion of policies outside computer systems, in a

sociological context. This may be because it is only in automated systems that it is possible to force such a simple dichotomy onto a naturally complicated world. There are approximately eight definitions of 'policy' in [12], with varying mixtures of flavour of imperative and authority. Most are biased towards imperatives, not all of which could be described as 'obligations'. Only one clearly emphasises authority. This book discusses policy conflicts entirely from the point of view of power struggles between opposing parties (often political parties) attempting to impose or frustrate the imposition of policies.

Star [13] uses a sociological approach to the subject. The concept of **due process** is used – the incorporation of differing viewpoints for decision-making in a fair and flexible manner. Sociological analysis of organisational problem solving in scientific communities yields the concept of **boundary objects** which are put forward as being a sufficiently plastic data structure for adopting differing viewpoints while maintaining continuity of identity. Clearly boundary objects have some similarity to the domains which we use for grouping objects to which policies refer, but there is not yet any evidence of how well they can be formalised.

6.3. Computer-Related Work

Bruggeman [14] discusses rights in an object-oriented environment, and analyses the conflicts which may arise when two elementary rights have the same object but different permission tags and the same priority. His 'basic conflict' corresponds to our positive-negative conflict of modalities, and he also notes the existence of 'latent conflicts', where the basic conflict has no immediate effect because of another right with a higher priority.

Wand & Woo [15] attempt to analyse conflict by interpreting it to mean a situation where it is impossible to resolve which of two possible stable states is to be chosen. This seems possibly suitable for the discussion of conflicts where two parties are putting forward different points of view and cannot decide between them, e.g. positive-negative conflicts. However, other conflicts are less suited to this approach.

Michael et al [16] address the problem of introducing precision into natural language security policies by treating them as a set of logical axioms and attempting to derive theorems from them. Policy conflicts are then represented by inconsistencies between the axioms. Many of the inconsistencies between the policies in a case study derived from the assumptions about real-world knowledge which are embedded in natural language. The conflicts were not formally categorised. The use of a theorem prover, similar in principle to the use of Prolog, appears a useful approach to the identification of possible conflicts in a large system.

An area which is related to policy conflicts, though not referring to it directly, is the use of contracts as a model for cooperation in distributed problem-solving. Smith & Davis [17] achieved task-sharing by using contracts, explicit agreements between nodes that generate a task (the manager) and nodes willing to execute the task (the contractor). The approach in ISTAR [18], 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.

6.4. Deontic Logic

The work which is closest in spirit to ours is in **deontic logics**, the logics of normative systems. These logics have operators which denote **obligation** and **permission**, either of states or actions. These operators correspond roughly to our own use of similar terms.

There are, however, differences of emphasis which have led us deliberately to retain different terminology. At a philosophical level, some people follow Kant's belief that 'ought' implies 'can' more or less strongly. Many flavours of deontic logic actually have the axiom that obligation implies permission, or even define them interchangeably, e.g. permission for an action *means* being not obliged to refrain from the action. By contrast, we wish to couple imperatives and authority rather loosely. It is obviously of interest when there is a imperatival policy for an unauthorised goal, but it is a common situation in human organisations for the goals to be set up before the resources to achieve them have been marshalled, e.g. when a company prospectus is issued.

Some more recent work overcomes these limitations. Wieringa et al [19] define obligation and permission independently; in their logic permission implies possibility, but does not imply obligation. Alchourron [20] also

defines obligation and permission independently; the possibility of obligation without permission is explicitly discussed, and characterised as 'inconsistent norming'. One could describe the equivalent situation for us as 'inconsistent policy-making'. Hage [21] deals with consistency of rules, which is closely related.

One issue which we are aware of, but have not resolved, is the logical status of policies. Alchourron makes the distinction between the logic of normative propositions and the logic of the norms themselves. As observed by Wieringa et al [22] there are two ways of interpreting 'It is forbidden to park here'. It may be the *observation* that a rule exists or the *promulgation* of the rule itself. One is a proposition with a truth value, while the other is a norm with the effect of a command. If we are to attempt to formulate a logic of policies – a task for the future – should it be a logic of propositions, of norms, or of both?

It is our intuition that the path of deontic logic is the correct one to follow if we wish to set up a theory of policies on a sound basis, but we cannot claim to have made much progress yet.

7. CONCLUSIONS

This paper has given an outline of our approach to the formalisation of management policies and then used this formalisation as a framework for the analysis of conflicts between policies. The different ways in which policy overlap can occur have been found to correspond closely to the informal intuitive classification of policies.

In the long run, automation of the detection and resolution of policy conflicts will be essential for effective automated distributed system management; if it were necessary to invoke human intervention for every potential or actual conflict, much of the benefit of automation would be lost. However, progress needs to be made on at least three fronts before this idea can be a reality.

First, a detailed knowledge of the application functions of distributed system management is needed. We have presented our examples in this paper on the basis of what is intuitive and familiar. Only when we know what actual conflicts arise can we be sure of the areas which need most attention.

Second, progress needs to be made on the generic formalisation of policies. It must be formal, because then it will be possible to treat it rationally, whether by creating simulation models or by reasoning in a formal logic. It must also be generic, because all kinds of policy have to fit into a compatible framework if conflicts between them are to be discussed.

Third, practical implementations of the theoretical model must be created and developed. Work has started on this in projects such as Domino [23] but there is a long way to go before it yields its benefits.

ACKNOWLEDGEMENTS

We acknowledge the contribution of colleagues in the Domino project in stimulating and criticising the ideas of this paper. Also, comments by John McDermid, Bret Michael and Philip Morris on earlier drafts have resulted in a number of improvements. We are grateful for the helpful suggestions of the anonymous reviewers. This work was carried out with the support of the UK DTI/SERC (Grant No. GR/F 35197).

REFERENCES

[1] Klerer, S.M., *The OSI Management Architecture: an Overview*. IEEE Network, 1988, vol. 2(2), pp. 20-29.

[2] Sloman, M.S. and J.D. Moffett, "Domain Management for Distributed Systems," in *Integrated Network Management I*, B. Meandzija and J. Westcott, Eds., 1989, North Holland, pp. 505-516.

[3] Moffett, J.D. and M.S. Sloman, "Delegation of Authority," in *Integrated Network Management II*, I. Krishnan and W. Zimmer, Eds., 1991, North Holland, pp. 595-606.

[4] Kanger, S., *Law and Logic*,Theoria, 1972, vol. 38, pp. 105-132.

[5] Heydon, A. and e. al, *Miro: Visual Specification of Security*. IEEE Transactions on Software Engineering, 1990, vol. 16(10), pp. 1185-1197.

[6] Varley, B., "User Administration and Accounting," in *Network and Distributed System Management*, M.S. Sloman and K. Kappel, Eds., 1993, Addison Wesley.

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

[8] Brewer, D.F.C. and M.J. Nash. "The Chinese Wall Security Policy," in *Proc. IEEE Symposium on Security and Privacy*. 1989, IEEE Computer Society Press.

[9] Nash, M.J. and K.R. Poland. "Some Conundrums Concerning Separation of Duty," in *Proc. IEEE Symposium on Security and Privacy*. 1990, IEEE Computer Society Press.

[10] Moffett, J.D., M.S. Sloman, and K.P. Twidle, *Specifying Discretionary Access Control Policy for Distributed Systems*. Computer Communications, 1990, vol. 13(9), pp. 571-580.

[11] Department of Defense (USA), *Department of Defense Trusted Computer System Evaluation Criteria*., Rep. DOD 5200.78 - STD, 1985.

[12] Hogwood, B. and L. Gunn, *Policy Analysis for the Real World*. 1990, Oxford University Press.

[13] Star, S.L., "The Structure of Ill-Structured Solutions: Boundary Objects and Heterogeneous Distributed Problem Solving," in *Distributed Artificial Intelligence 2*, M. Huhns & Casser L., Eds. 1989, London: Pitman, pp. 37- 54.

[14] Bruggemann, H.H. "Rights in an Object-Oriented Environment," in *Proc. IFIP WG 11.3 Fifth Working Conference on Database Security*. 1991, Sheperdstown, WV.

[15] Wand, Y. and C. Woo, *A Formal Model for Analysing Organizational Computing Concepts*. 1992, Faculty of Commerce and Business Administration, University of British Columbia, Vancouver, Canada.

[16] Michael, J.B., *et al*. "On the Axiomatization of Security Policy," in *Proc. IFIP WG 11.3 Sixth Working Conference on Database Security*, 1992, Vancouver, Canada.

[17] Smith, R.G. and R. Davis, *Frameworks for Cooperation in Distributed Problem, Solving*. IEEE Transactions on Systems, Man and Cybernetics, 1981, vol. SMC-11(1), pp. 61-70.

[18] Dowson, M., *ISTAR - An Integrated Project Support Environment*. J SIGPLAN Notices, 1987, vol. 22(1), pp. 27 - 33.

[19] Wieringa, R., *et al*., *The Inheritance of Dynamic and Deontic Integrity Constraints*. Annals of Mathematics and Artificial Intelligence, 1991, vol. 3, pp. 393-428.

[20] Alchourron, C.E. "Philosophical Foundations of Deontic Logic and its Practical Applications in Computational Contexts," in *Proc. First International Workshop on Deontic Logic in Computer Science (DEON'91)*, 1991, Amsterdam, The Netherlands: Free University of Amsterdam.

[21] Hage, J. "Consistency of Rules," in *Proc. First International Workshop on Deontic Logic in Computer Science (DEON'91)*, 1991, Amsterdam, The Netherlands: Free University of Amsterdam.

[22] Wieringa, R., J.-J. Meyer, and H. Weigand, "Specifying Dynamic and Deontic Integrity Constraints," in *Data and Knowledge Engineering 4*, 1989, North Holland, pp. 157-189.

[23] Sloman, M.S., J.D. Moffett, and K.P. Twidle, *Domino Domains and Policies: An Introduction to the Project Results*, 1992, Dept of Computing, Imperial College, University of London.