The Policy Obstacle Course:
The Realisation of Low-Level Policies in Distributed Computer Systems

SCHEMA/York/93/1.2 15 July, 1994

Jonathan D. Moffett[a] , Dirk Jonscher[b], John A. McDermid[a]

[a] University of York, Department of Computer Science,
Heslington, York Y01 5DD, UK
{ jdm, jam }@minster.york.ac.uk

[b] Universität Zürich, Institut für Informatik,
Winterthurerstr. 190, CH - 8057 Zürich, Switzerland
jonscher@ifi.unizh.ch

Short title: **The Policy Obstacle Course**

Abstract

The paper proposes a framework for the realisation of low-level policies which are embedded within distributed computer systems, i.e. those which are intended to influence system actions. The framework arises naturally from the nature of distributed systems: obligation policies are interpreted by a Situation Monitor which decides when to initiate operation messages; while authority are used by Reference Monitors in deciding whether to permit operations to be performed.

All low-level policies, except those whose function is to propose an action, act as filters (or obstacles), and the process that an action goes through before reaching its target object may be described as an obstacle course, with filter policies being the obstacles which must be hurdled. The different considerations which affect human and automated implementation of policies are introduced. The framework is used to describe certain classes of policy conflicts and their resolution. The implementation issues which arise in a distributed system are discussed.

1. INTRODUCTION

The purpose of this paper is to describe a framework for the realisation of low-level policies which are embedded within a distributed computer system. This arises naturally from the message-passing environment of distributed systems. It is a convenient framework for system management policies, and is potentially useful for the realisation of safety policies in safety-critical systems.

1.1 Background and Motivation

The Need for Policies in Systems

One of the most notable advances in computer systems during the last few decades is their progression from batch information-processing to support for the control of real-world systems, which vary from those which handle information, such as telecommunications systems, to electro-mechanical systems such as power plants. All these computer systems, regardless of which end of the spectrum they are at, have one thing in common; they provide support for the control of the real-world system within which they are embedded. The boundaries of a real-world system of this kind include the human beings who take part in its operation, in addition to the hardware of the system.

The way in which a controller, whether human or automated, determines the appropriate control actions is by reference to the policies which apply to the system. For some systems these policies may not be represented explicitly within the system itself. A human controller may have intuitive knowledge about what is required; while if the control is automated, the control program may embody the requirements of the system's policies directly in its code. However, there are advantages in their explicit representation as embedded policies, because this enables the decision-making part of the control loop to be separated from monitoring and control actions. The result of this is increased visibility of the policies and ease of their maintenance.

The explicit representation of policies is particularly important for distributed systems in which there may be a number of enterprises participating. Each may have a different view of the appropriate policies which should be in force, and it may be necessary for them to negotiate a set of policies which are acceptable to all of them.

Policy Hierarchies

It is well understood that policies are typically first stated at a high level, and then progressively decomposed through intermediate policies down to a point where procedures, consisting of atomic actions, are specified which will, taken together, realise the high-level policy [Moffett 1993a]. Atomic actions are, in this context, computer system operations or human actions which do not require any further decomposition or explication. We refer to the combination of an agent, a modality (obligation or permission, or their negations) and a procedure consisting of atomic actions, as a low-level policy.

A brief example illustrates a policy hierarchy. There is a power plant, for which one of the high-level policies is:

The manager of the power plant is to ensure its safe operation P1

One of its components is a boiler, which is known to be safe when its temperature is less than 100°C. It is potentially unsafe above 100°C. Therefore one of the intermediate policies which support policy P1 is:

The manager is to ensure that the temperature of the boiler remains below 100°C P2

A number of factors affect the temperature of the boiler. They include:

- The Close_Valve operation causes the temperature to increase;
- The Open_Valve operation causes the temperature to decrease;
- Influences from the external environment may affect the temperature;
- Faults in the system may affect the temperature.

Clearly, in order to ensure the success of policy P2, a full analysis of the system and its environment is needed before defining a set of procedures which will cover all feasible eventualities. However, for the purpose of this example we set out two low-level policies which will help to achieve P2:

When Temp > 100°C and the valve is not Open, the Operator shall initiate Open_Valve P3

All agents are prohibited from Close_Valve if Temp > 100° C P4

A number of issues are raised by this simple example:

a) How do we provide a sufficiently full description of the system and its environment in order to carry out the analysis of the system's behaviour, in both correct and faulty modes? This is an area of software engineering which has been widely researched.

b) How are we to decompose from high-level policies such as P1 to low-level policies such as P3 and P4 so as to ensure that P1 is satisfied? This has been approached from several directions, including [Moffett 1993a, McDermid 1994]. In our example, we should note that this is complicated by other high-level goals such as those relating to efficiency, which will interact with the safety policies.

c) At what point do we move from off-line to on-line decomposition? It must not be necessary for the (human or automated) controller to decompose right from the top of the policy hierarchy every time the system runs; there will be a statically-generated set of intermediate policies which are available for run-time use. But it may be useful to have some intermediate policies which are interpreted on-line and decomposed into low-level policies. For example, there may be trade-offs that can be made at run-time in order to maximise efficiency while remaining safe, resulting in different procedures in different circumstances.

A half-way house between static off-line analysis and dynamic on-line interpretation is the possibility of defining different modes in which a system operates, and applying different low-level policies accordingly. Possible modes for the boiler are Starting-up, Running, and Shutting-down, with a different set of low-level policies for each.

Static off-line decomposition of policies is a software engineering activity. On-line interpretation and decomposition of policies, whether by an automated agent or a human, needs expert system or other AI techniques [Leitch 1993, Nordvik 1993].

All of the above issues are important, and require further work on them. However, the chief theme of this paper is a different issue: *What are the possible forms of low-level policies*? It is our contention that the object-based model of systems, which is common, perhaps universal, in distributed systems, dictates a particular framework for the realisation of low-level policies. From this framework there arises a natural treatment of the description and resolution of certain kinds of policy conflict.

1.2 Computer System Assumptions

We make two assumptions about the computer systems which we are discussing: that they are object-based; and that their operation and management is often a human/automated hybrid. The first of these assumptions dictates a clear-cut framework for low-level policies, and the second then blurs this somewhat.

Object-Based View

We take a simple object-based view of the world. Actions are performed when an **agent object** sends an **operation message** to a **target object**, which will perform the operation provided that it is in its exported interface. The message may have parameters associated with it. This familiar view is illustrated in Figure 1.

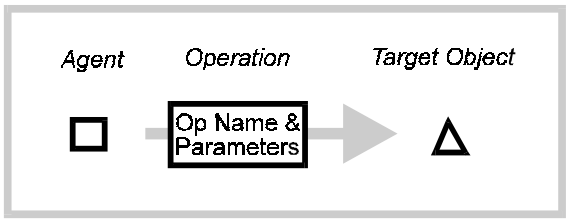

Figure 1 Agent, Operation and Target Object

This view has important consequences for the realisation of low-level policies in computer systems: there are only two ways in which it is possible to influence whether an operation is performed upon a target object. One is by influencing whether the operation is **initiated**, i.e. a message for the target is emitted from the agent object. The other is by influencing whether the operation message, once initiated, reaches its target object. There is therefore a natural division of low-level policies into two kinds. Policies which influence the obligation to initiate an operation are called **obligation policies**, for example policy P3 above; and those which influence whether an initiated operation is actually performed upon its target object are called **authority policies**, for example policy P4. Obligation policies are concerned with **initiating,**

or **inhibiting** the initiation of, actions; and with positive and negative **freedoms** which override initiation and inhibition policies. Authority policies are to do with the possession of legitimate power. They are concerned with **permitting** or **prohibiting** actions[1]. In this situation authority is essentially a negative form of power; the authority which resides in computer system mechanisms can prevent actions from taking place, but it cannot ensure that they succeed.

Hybrid Human/Automated Computer Systems

Distributed computer systems, particularly embedded systems, are complex, and the actions needed to control and manage them are complex also. Some forms of complexity are best dealt with by computer systems, e.g. a complex shut-down procedure for a power plant, while others, particularly where the problem domain has not been completely mapped out, are best dealt with by human beings. So, although it might be our goal to enable as much automation as possible, it must be assumed that for the foreseeable future the operation and management of systems will involve both human and automated agents. The effect of this on low-level policies must therefore be considered.

It is usual to refer to humans performing actions, and automated agents performing operations but we do not make a rigid distinction between them. For clarity we use "operation" to identify what is to be done, analogously to "method" in object-oriented programming, and "action" to denote the tuple of agent, target object and operation.

1.3 Paper Contents

The remainder of the paper is structured as follows. Section 2 describes the major components of the framework: obligation and authority policies; the Situation Monitor to interpret and enact obligation policies; and the Reference Monitor to interpret and enforce authority policies. Section 3 describes the analysis and resolution of some common kinds of policy conflict. Sections 4 discusses implementation issues, particularly those related to distribution. Finally, related work is described in section 5 and section 6 draws out the paper's conclusions.

---

[1] We use "authority" as either an adjective or a noun. The other candidate word was "authorisation", which may either mean "conferring authority upon" or the process of determining whether an action is permitted, and so we avoid its use if there is a risk of ambiguity. We rejected "power" because we are only addressing the legitimate use of power, and not its misuse, such as power gained by breaking into a system. It should be noted that the meanings which we attach to the terms used in this framework should be precisely interpreted according to the mechanisms which we are describing, and should not be assumed to extend to informal English usage.

2. POLICY FRAMEWORK

2.1 Outline

We take a very wide general view of the possible meanings of "policy"; it is any statement which is intended to have a future influence on actions in a system, or on its state. Many policies are very broad, high-level statements about system states. As indicated above we concentrate here on low-level policies which are intended to influence actions, and which fit into an object-based view of the world.

A first view of low-level policies may be that they are mainly positive in nature, concerned with initiating or permitting actions. However, when one looks at them, as expressed by humans in policy manuals and statements, they are as much concerned with preventing actions, recovering from unwanted actions, excluding people or objects from their scope, overriding them with other polices and stating limitations for their applicability, as with achieving positive actions.

We propose the following picture of the relationship between policies and actions. One way in which an action may be initiated is when an event triggers an **initiation policy**, i.e. one which proposes a candidate action to be initiated. However, the action will only be initiated if it is not prevented by any negative obligation policies which filter it out, i.e. if it is not **inhibited**. Even after it has been initiated it will not operate on its target object unless it is permitted to do so by the authority policies which apply to it. Note that in our model all kinds of policy except a positive obligation policy have negative or restrictive characteristics; they are concerned either with filtering out actions, or preventing that filter in order to allow an action. We refer to all kinds of policy except simple positive obligation policies as **filter policies**.

The whole process of initiating actions, and then filtering them through other policies can be regarded as an obstacle course which has to be surmounted before the action is performed. See Figure 2. An agent may be inhibited from initiating a candidate action, or prohibited from performing an initiated action, or may succeed in overcoming both these obstacles and perform the action on the target. We call the mechanism which interprets and enacts obligation policies the **Situation Monitor**, which has the function of deciding when agents are to initiate operations on target objects. The **Reference Monitor** [Anderson 1972] interprets authority policies to decide whether an already initiated action is permitted, and enforces that decision by rejecting the operation message if it is prohibited (or perhaps modifying it, in the case of database queries [Stonebraker 1975]).

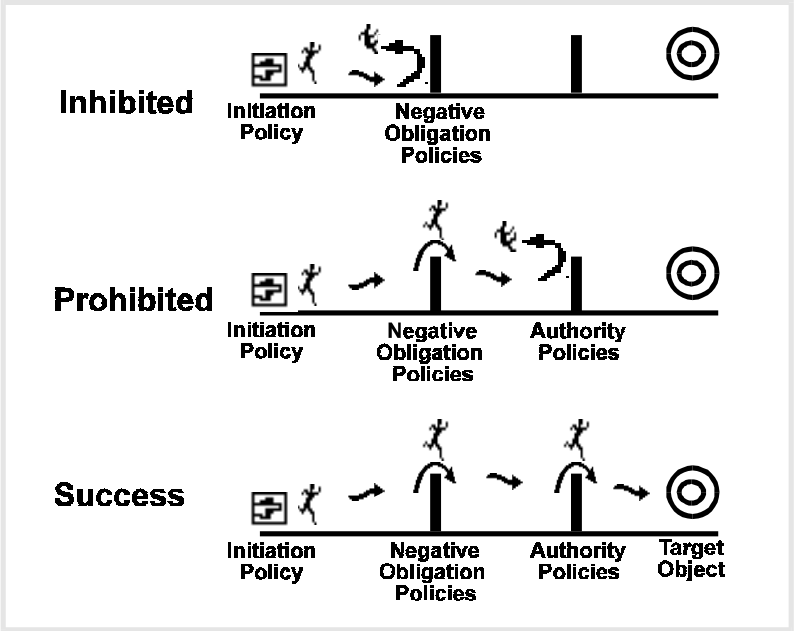

Figure 2 The Policy Obstacle Course

2.2 Obligation and Authority Policies

We show a coarse-grained picture of our view of our world in Figure 3, assuming that the agent is automated, and that the Situation Monitor is incorporated within the agent. We assume that there are databases of obligation and authority policies which are available to the monitors.

The situation monitor is a proactive mechanism which, on the basis of its obligation policies, monitors relevant events in the system and causes the agent to initiate an operation request message when appropriate. Automated agents have to be trusted to initiate operation requests reliably when required to by their situation monitors.

The reference monitor is passive; operation requests are directed to it and it evaluates them when they arrive. It decides, on the basis of its authority policies, whether to permit or prohibit the request. There may be a number of reference monitors in different domains of a distributed system, between the initiating agent and the target object, for example one in the agent's domain, one on the network, and one in the target object's domain. If an operation request is permitted by all the reference monitors in the path of the message, the operation will be performed by the target object.

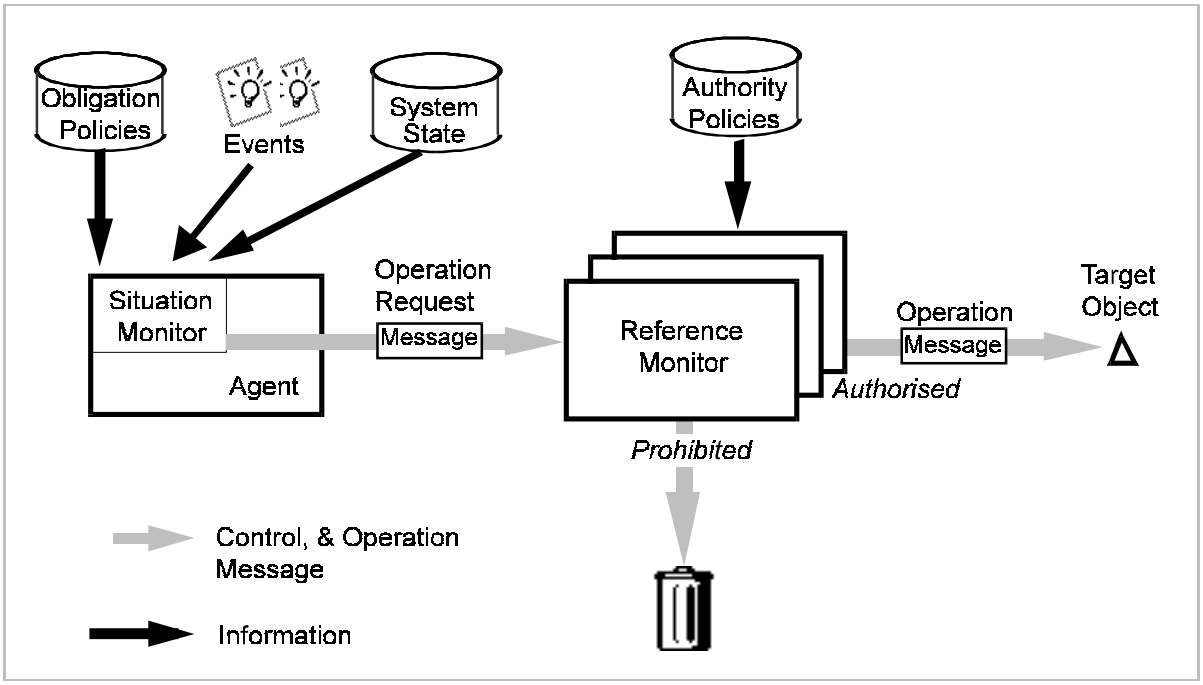

Figure 3 A Coarse-Grained View of Policies (Automated Agents)

2.3 The Situation Monitor

The Situation Monitor is split into two parts. The proactive part monitors events in the system. It determines, on the basis of these events and the **initiation policies** that are in its database, when there is an action which is a candidate to be initiated. The reactive part then filters out any action which is to be suppressed as a result of the negative obligation policies in the database. For an automated agent, if the candidate action passes successfully through the filter, it is initiated by the agent, see Figure 4.

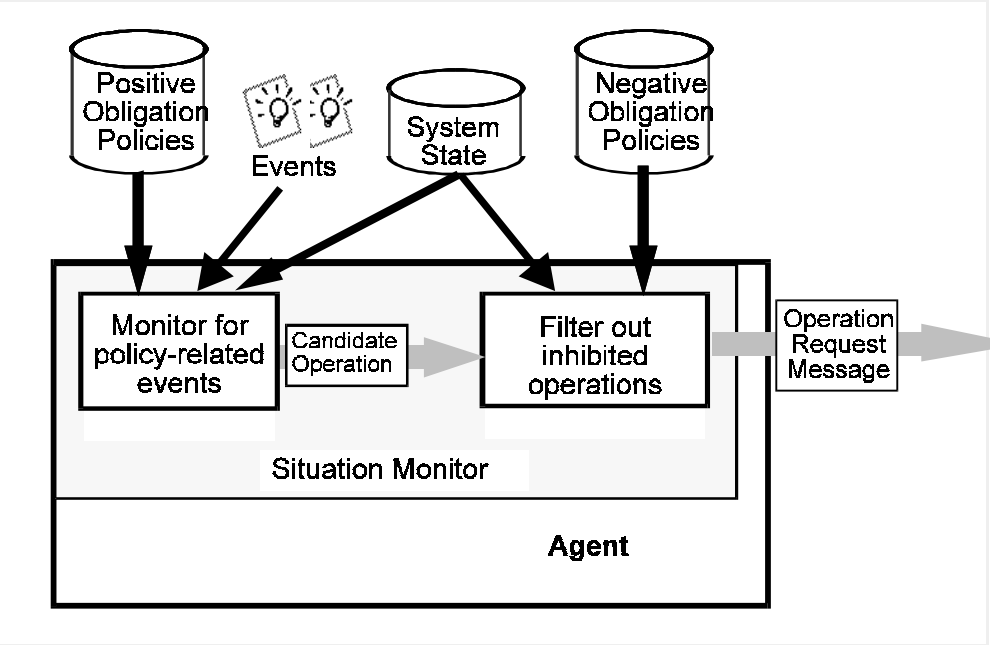

Figure 4 Situation Monitor (Automated Agents)

On the other hand, if the agent who is to initiate it is human, then there can be no such automatic initiation. The system will send a message to the agent's screen (or other output device) requesting him/her to confirm or start the action. An example of this kind of interaction is:

- An instruction by a nuclear power plant control system to the operator, in a failure situation, to switch from one coolant pumping system to another. In many power plants, as a matter of fundamental policy, the operator is always "in the loop" and can refuse to do what the system recommends. This is an example of human resolution of a negative freedom, and is discussed further in section 3.1.

It will not be a system error if the human does not comply, with the following general alternatives to compliance: the request is ignored; the human replies, refusing; or the human performs a different action from the one which was requested. Subsequently the system might monitor the fulfilment of an initiation policy and in case of non-fulfilment initiate a contingency action automatically, see [Jonscher 1992] for further discussion.

There may be many reasons for a human not complying with the system's request, including policies which apply to the human but which are not held in the system's database of policies. Typically these will be in the form of policy or procedures manuals which the human will use in deciding whether to comply with the system's request. Also, in safety and other situations, the human may have access to different state information from the system, and thus determine that a different action is needed. The issue of the balance of control between man and machine is also relevant. The situation is altered from Figure 4, by the addition of a further set of policies which are directly interpreted by the human agent. This is shown in Figure 5.

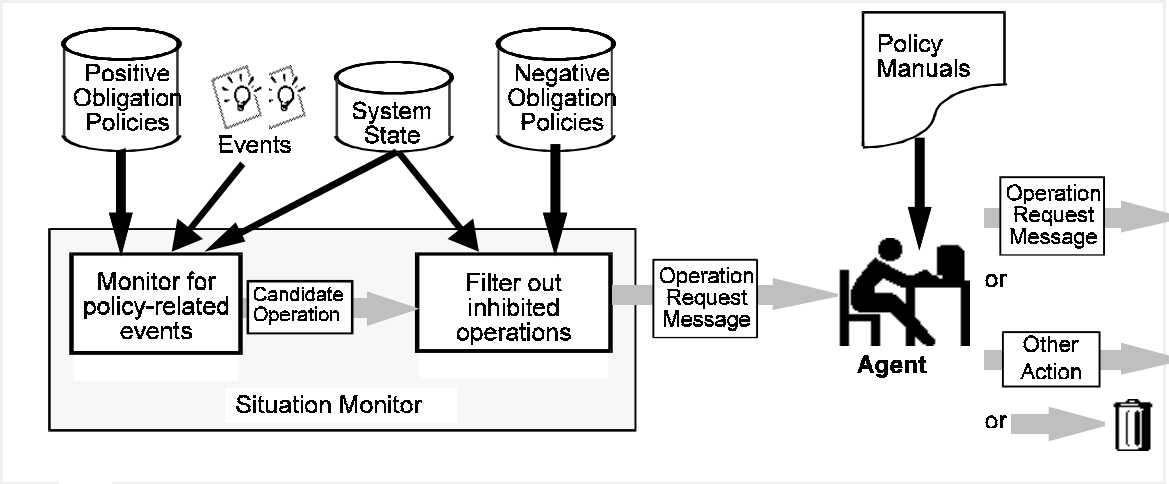

Figure 5 Situation Monitor with a Human Agent

Another scenario is that the human proposes the action, whether spontaneously or through encountering an event and obeying the policy and procedures manuals. The relevant event which triggers the proposal for an action is outside the computer system, and there may also be some relevant policies in the manuals which the human has also taken into account before proposing the action. The computer system may institute a dialogue with the human, requesting confirmation of the action before initiating it, for example:

- A pilot proposes to shut down one engine during flight when there are no failure indications. The computer system requests confirmation from the pilot before doing so.

A dialogue of this kind does not fit comfortably within our framework; the investigation of how best to reconcile humans with automated policies is an area for further research.

2.4 Reference Monitor

The role of the Reference Monitor is to search its authority policies database for authority policies which apply to an operation request which it has intercepted; to determine whether the result of evaluating them should be to permit or prohibit the action; and to enforce that decision. For authority policies, human interface issues do not arise for operations within the computer system. There is no external interface unless there is a requirement to log the happening or inform a human, typically because the action was prohibited. This is the case whether it was initiated by an automated or a human agent.

2.5 Policy Modes and Structure

We have two main modes of policy: obligation and authority. In addition, there may be negative aspects of policies. However, there is a variety of possibilities for negation in policies, rather than one single form of negative policy.

Modes of Obligation Policy

We start with the Initiation (positive obligation) policy:

1 (*Agent*) A is **obliged** to initiate (*operation*) B on (*target object*) C.

The intent of this is to prompt the **initiation** of the operation.

There are three potentially useful kinds of negative obligation policy to which we assign descriptions "inhibition", "freedom" and "negative freedom" as follows:

2 A is **inhibited** from initiating B on C (derived from: A is commanded *not* to initiate B on C).

3 A is **free** to initiate B on C (derived from: A is *not* commanded *not* to initiate B on C).

The intent of this is to nullify any inhibition of the operation [positive freedom].

4 A is **free not** to initiate B on C (derived from: A is *not* commanded to initiate B on C) [negative freedom].

We believe that the main value of freedoms will be if humans are in the loop, as shown in section 2.3, although one can envisage some applications where it may be useful to define freedoms for automated agents.

There is also the possibility of including negative actions in policies. What might we mean by doing a not-action, as opposed to not doing an action? There are several possible interpretations of not-actions. We illustrate them against this example: "B is commanded to not-increase D's salary by $5000."

- [Wieringa 1991] defines a not-action simply as the non-performance of the action. B will do nothing.
- [Khosla 1988] interprets it to mean doing anything except the named action. B will do anything at all except increase D's salary.
- We could define a not-action as a counter-action which recovers from the effect of an action which has already taken place. In this case B will remove D's salary increase, rather than decreasing the salary.

Counter-actions for recovery may be important for both safety- and security-critical systems, for handling violations of requirements. In general, negative actions, particularly counter-actions, are interesting as an approach to handling operation failures, but their complication is such that we are not attempting to integrate them into our view of obligation policies at this stage.

Modes of Authority Policy

There are only two useful modes of authority policy. Both of them are used by the reference monitor to decide about a previously initiated operation request. The positive authority policy is:

5 (*Agent*) A is **permitted** to perform (*operation*) B on (*target object*) C.

Its negation is:

6 A is **prohibited** to perform B on C (derived from: A is *not* permitted to perform B on C).

Policy Structure

Policies are given a fixed structure, in order to enable their manipulation as system objects, as discussed in [Moffett 1994a], However, initiation policies (positive obligation policies) have a different structure from all other kinds of policy, i.e. negative obligation policies, and authority policies.

The purpose of initiation policies is to propose actions. An **action** is a tuple consisting of an agent (often the agent incorporating the Situation Monitor), an operation (with parameters) and a target object. The intuitive meaning of an initiation policy is that it is an instruction to the Situation Monitor to propose an action, when an event occurs, e.g.:

- Every Friday, the Operator is to start the backup job on the department's files.

However, in many cases an event is intended to initiate multiple actions, e.g.:

- Every Friday, all the members of the department are to start the backup job on their own files.

Therefore our general view of an initiation policy is that it is an event and a set of actions. The Situation Monitor monitors proactively for events and proposes the actions defined in the corresponding initiation policies.

Our general view of all other policies is that they are rules which are interpreted by the Monitor in deciding whether to inhibit/authorise/prohibit, etc., any action which is matched by the policy. Their structure is determined by the requirement that they have to be able to match actions. Their attributes therefore include **Agents**, **Operations** and **Target Objects**. In addition, there are five different **Modes** (numbers 2 - 6, discussed above), and a policy may also have a **Constraint** attribute to limit its applicability, which is related to the system state, e.g. phase of flight for an aircraft, or time of day. For example a policy might only apply only outside normal working hours.

3. POLICY CONFLICTS

We discuss here those conflicts which can be described directly in terms of the framework which we are proposing. Note that this is only a small portion of the various kinds of conflict which can arise between policies; other kinds of conflict have been classified and discussed at more length in [Moffett 1994b].

3.1 Conflicts between Obligation Policies

If there is an action proposed by an initiation policy and, in addition, one or more negative obligation policies which match it, there is a conflict about whether the action should actually be initiated. We set out in Table 1 the various possibilities which can occur in this case. These only relate to obligation policies taken by themselves; we discuss below (section 3.2) their possible interactions with authority policies.

The table lists the results which arise from the eight possible combinations of modes of policy. Multiple matchings by the same mode of policy are ignored, i.e. two Inhibiting policies have

the same effect as one. The third column tabulates the outcome, which may be: the action is Initiated; or the action is Inhibited (i.e. not Initiated); or the result is Indeterminate (see rule 4).

Rule	Modes of Obligation Policy which Match Proposed Action	Effect on Proposed Action
1.	None	Initiated
2.	Inhibiting	Inhibited
3.	Free to	Initiated
4.	Free not to	Indeterminate
5.	Inhibiting & Free to	Initiated
6.	Inhibiting & Free not to	Inhibited
7.	Free to & Free not to	Indeterminate
8.	Inhibiting & Free to & Free not to	Indeterminate

Table 1 Rules for Policy Modes and Actions

Our reasons for formulating these rules are as follows:

1. If there are no negative obligation policies matching a candidate action proposed by an initiation policy, then it is initiated.
2. If there is (only) an inhibiting policy matching the candidate action, then the action is inhibited. Given rule 1, this is the only way in which the rules could be formulated to allow candidate actions to be inhibited.
3. As for rule 1, since there is no reason for a positive freedom to interfere with a candidate action.
4. The purpose of a negative freedom (free not to) is to allow a candidate action to be overridden by the agent. Where a candidate action is accompanied by a negative freedom, the obligation to carry out the candidate action is removed. There is now no positive or negative obligation so, at the system level, the result is indeterminate but the agent must be informed. The issue must be resolved at agent level.

5 The purpose of a positive freedom (free to) is to allow an Inhibition to be overridden. Where a positive freedom and an Inhibition occur together, the inhibition is "cancelled out" and the action is initiated.

6. As for rule 2, since there is no reason for a negative freedom to interfere with a inhibition.
7. As for rule 4, since the positive freedom has no effect on a candidate action.
8. As for rule 4, since the positive freedom annuls the inhibition, and we are left with "freedom not to".

The proposed rules are straightforward with respect to inhibition and positive freedom. Negative freedoms, however, introduce indeterminacy at the system level. The Situation Monitor must inform the agent, which can resolve whether or not the action is to be initiated, based on an application-level view of the candidate action. This resolution may either be done by reference to a human agent at run-time or by resolving it at design time in the application.

Negative freedoms are a useful means of representing, in automated policies, the need to include a human in the decision-making process. An example of human resolution of a negative freedom was given in section 2.3: the operator is free to refuse to switch from one coolant pumping system to another in the power plant.

3.2 Other conflicts

Conflicts between Authority Policies

Conflicts between authority policies are much simpler than for obligation policies, because there are only two modes. The rules we use depend upon the default policy, or closure assumption. It will have been noted that no mention was made of a default policy for obligation policies, since no action even becomes a candidate unless there is an initiation policy to generate it. On the other hand the question of default authority policies is a real one, since an initiated action may be intercepted by a reference monitor which has no matching policy, either permission or prohibition, for the action.

This subject has been discussed at some length within the security community. We use a simple working principle. If the default authority policy is prohibition – the common case of a "closed world assumption" – then explicit Prohibition has priority over explicit Permission, which has priority over the default Prohibition. In the converse case, where the default is Permission, explicit Permission has priority over explicit Prohibition, which has priority over the default Permission. Permission as the default policy is of course unfashionable but if security barriers, like hem-lines, are raised at some time in the future, our model of policies will cope with the New Look.

Conflicts between Monitors

There is the possibility that an action is initiated by the Situation Monitor but prohibited by a Reference Monitor, or permitted by one Reference Monitor but prohibited by another (see section 4.4). This may be considered to be a conflict. The result is clear: the Reference Monitor's prohibition halts the operation message's progress, and prevents the action from being performed. It may be a problem that agents are obliged to do something for which they have not got the authority, and on the face of it, this is an inconsistent specification. In many cases it is necessary to identify and resolve such conflicts at design time.

4. IMPLEMENTATION ISSUES IN A DISTRIBUTED ENVIRONMENT

For relative brevity, we have so far taken a simple view of the individual components of policies, but have made implicit assumptions as a result. In this section we discuss the components of policies in more detail, and the implications for their realisation in a distributed environment.

4.1 Events

Events are assumed to be generated by a variety of components of the system, and signalled to the Situation Monitor which, whenever an event occurs, selects the corresponding candidate actions. We envisage that many events will be simple, e.g. timer events. There may also be initiation policies in the database whose events contain parameters which are bound to actual values by the occurrence of an event. For example, there may be an initiation policy which is intended to initiate a recovery action in the event of failure, e.g.

- When (line *X* fails) switch *X*'s traffic to the alternative line for *X*.[2];

The actual value of *X*, and therefore the actual object on which to perform the action, are not determined in the policy. The Situation Monitor has to determine which is the "alternative line" for the line which is bound to *X*

In general we would expect events to be either the arrival of signals from objects, or state transitions over system state and time. In the latter case, it is necessary for the state to be accessible to the Situation Monitor; see the discussion in the next section.

4.2 Predicates in Distributed Systems Policies

Predicates may appear in policies, to determine whether they are to apply when a relevant event occurs or an action is presented to a monitor. These predicates either determine membership of the agent or target object sets or are separately expressed as a constraint in the policy.

We can distinguish the following kinds of information which may be needed by policy predicates, and we discuss how easily it can be made available:

a) Parameters of the operation message, e.g. the amount of a financial transaction, are immediately available to the monitors (provided the information is not encrypted).

b) System variables, such as local time, can be assumed to be readily available to the monitors. These do not present any problem in policy evaluation in distributed systems, although they give rise to problems of defining the requirements for policies; if access to a European database from a terminal in the USA is to be permitted "during working hours", then whose working hours (and why)?

[2] Clearly, recovery action of this kind cannot be carried out by a simple action. A transaction is needed which includes both identifying the alternative line and switching traffic to it.

c) Properties of agents which can be authenticated and attached to the operation request, e.g. the names of the agent's ancestor domains may be attached to the operation request [Sloman 1994]. Provided it is authenticated by a trusted party, this information can be used by a monitor without the overhead of having to query the object.

d) Properties of agents and target objects which are available through queries on the system, e.g. the terminal at which the agent is currently logged on, may be locally available to the situation monitor.

e) Attributes of objects which could be ascertained by querying the object, e.g. a policy whose target object set is "all black cars". It is impossible to know, without querying its attributes, whether the policy applies to a particular car. However, in a distributed system this may be impossible, because the object is in a domain that the monitor is not trusted to query. Even if it is possible for the monitor to query the object, this may result in an unacceptable performance penalty.

Some of these predicates, i.e. operation parameters and system variables, clearly belong in the Constraints attribute of a policy. Other predicates could be expressed as constraints or within the definition of an object set, e.g. the policy applying to "all black cars", which could instead have a target object set of "all cars", with the constraint "provided they are black". We suggest that a pragmatic approach is to specify in the relevant object set any predicates on the attributes of agents and target objects, and specify other predicates in the policy constraint.

Note that negative predicates are often essential. Suppose we wish to express the policy "no agent is permitted to Read the security database unless he has the Security attribute", and assume that prohibition is the default. The policy has to be expressed as follows: "all agents which do not possess the Security attribute (*agent set with predicate*) are prohibited (*mode*) to Read (*operation*) the security database (*target object set*)". Prohibition is combined with a negative predicate "does *not* possess the Security attribute". Positive authority combined with a positive predicate, i.e. permission together with "possesses the Security attribute" would not be satisfactory, because some other permission may have to be satisfied also; the policy is necessary but perhaps not sufficient.

4.3 Agents

The agents and target objects of policies are typically specified as sets. This applies not only to filter policies, where a set of objects may be specified against which to match the generated action, but also to the agents of Initiation Policies, where there may be the need to identify multiple agents because events may generate multiple candidate actions, e.g.:

- Every Friday at 12 noon, all the members of the department are to start the backup job on their own files.

The event here is simple, but the Situation Monitor has the task of identifying all department members in order to generate a candidate action for each one. The members of a set may be defined by enumeration or by predicate. A mechanism such as the use of Domains [Sloman 1994] overcomes the problem of scalability in set enumeration. There is no scalability problem in defining set membership by predicate but, as discussed above, in distributed

systems there can be a problem of evaluating the predicate if it refers to information which is not globally available.

A potential problem arises in the case where a group of agents co-operate in performing tasks. Only one agent is to perform the action, but the obligation is aimed at the group as a whole. Up to this point we have assumed that the Situation Monitor is incorporated within the agent, but in this case it has to be a separate object from the agents which it is serving. There are two possible approaches to ensuring that a single agent initiates the action. The solution chosen will depend upon the style of operation of the group.

- The Situation Monitor generates the action for a single known agent which delegates the task to other agents;
- The action is proposed for every member of the group, which then needs a co-operation protocol, e.g. setting a semaphore, to resolve which performs it.

4.4 Distributed Policies

Another aspect of distributed systems is that policies may be distributed through the system, and that only some of the policies "belong" to a particular monitor. It is necessary to allow for the binding of monitors to particular policy information. For example, Figure 6 shows three reference monitors, local to the agent, the network and the target object. There could in principle be more than this if the agent and target domains are on different networks, each performing its own access control. When there are multiple reference monitors, the function of each is identical: to intercept an operation message and only allow it to pass onwards if it is authorised in that domain. In many systems there is just a single reference monitor, in the domain of the target object.

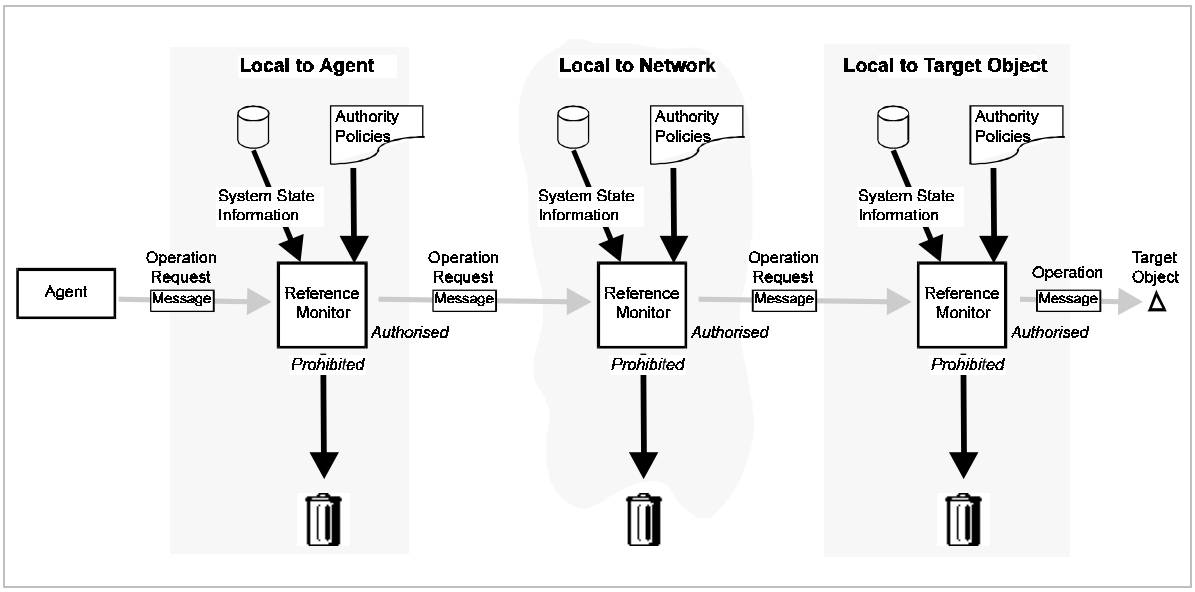

Figure 6 Multiple Reference Monitors

Typically, however, if there are multiple monitors each will be under the control of a separate, independent enterprise, with separate, possibly contradictory policies. A computer bureau

might have a policy to permit all accesses from a client (because it can charge for them), while the client may have a policy of forbidding access to the bureau except for its senior staff. In this situation the client's reference monitor, local to the client, would be using quite different policies from the bureau's reference monitor, local to the bureau. It is therefore necessary to allow for the binding of monitors to particular policy information.

5. RELATED WORK

The earlier published work relating to policies and computer systems concentrates on providing automated support for humans who are interpreting policies. Dobson [1989] although dealing with the processes of system development, proposed a framework which would provide a language for describing security policies. [Bedford-Roberts 1991] describes a system for capturing corporate policies which can be queried by users. [Storrs 1991] describes applications for supporting policy-makers in the UK Department of Social Security. [Nguyen 1993] describes a structured business policy model, focusing on IT activities, for the capturing and understanding of business-IT changes and choices.

Work by Sibley, Michael and others are mainly in the same spirit, e.g. [Sibley 1993]. However, Michael [1991]. proposed a RWA (Ready, Willing and Able) paradigm in which being Willing corresponds to a positive obligation, while being Able corresponds to positive authority. "Ready" appears to have been more problematic. This is not surprising, since obligation/willing and able/authority correspond quite closely to the *obligation* and *permission* modes of deontic logic; see for example [Wieringa 1991] for an approach to similar problems by formulating them in a deontic logic. We have considered using deontic logic, but its technical problems do not yet appear to have been resolved sufficiently to make this feasible.

[Masullo 1993] proposes a Policy Management System which is capable of interpretation by the system itself, as well as by humans. Its components would be a language for the specification of policy functions, an information manager, and a method specification language for defining active processes.

6. CONCLUSIONS AND FUTURE WORK

This paper has described a framework for the realisation of policies that are embedded in computer systems, which follows naturally from the message-passing paradigm which is universal in distributed systems. There are two classes of policy in the framework: obligation policies which are concerned with the initiation of operation messages, and authority policies which are concerned with their subsequent authorisation. It also emerges that positive obligation policies (initiation policies) are different in nature from all others in the framework. They are proactive policies which the situation monitor uses to initiate actions, while the remaining kinds of policy are reactive to the action proposed by the initiation policy.

The framework throws some light on conflict identification and resolution – the rules for resolution of certain kinds of conflict arise naturally from the framework and are described in Table 1. It also clarifies the distinction between local and remote policies, and so identifies the kinds of policy which may be practically applied to local and remote objects.

However, discussion of our framework has also highlighted many areas which require further work.

The framework deals entirely with low-level policies concerned with atomic actions. Work is also required on higher-level policies, and policies which are expressed in terms of state, rather than actions. In particular, if we are to make a useful contribution to managing safety policies, we must come to terms with the fact that many are expressed in terms of avoidance of undesired states [Moffett 1994c], which are inherently at an intermediate level. Also, it is necessary to come to terms with the many cases in which policies are compromised or violated but, as with ordinary human policies, it is still necessary to continue to manage and operate.

First, we have concentrated on what is required to enable an action to reach the point of being performed. However, operations may fail at many points. A candidate action may be proposed as a result of an initiation policy, but inhibited; it may be passed to the agent for initiation but the agent may fail to do so; it may be initiated by the agent but prohibited by a reference monitor. How should the system react to these situations? An operation may also fail after it has started to be executed. Possible reasons include inadequate resources to complete the action, and the failure of cascaded or transitive messages which are generated by the target object. The possible reasons for such failures, and the way in which the system should deal them, require to be investigated.

The work done so far has concentrated on relatively simple aspects of policies: the initiation of actions and decisions about whether or not they are to be permitted. Work in the database field [Larrondo-Petrie 1990; Chakravarthy 1989] has shown that a more complicated view will have to be taken in order to deal with realistic situations. In particular, the criterion for initiation of actions cannot be expressed by a single condition but may be the result of a number of conditions, both positive and negative in nature.

The test of this model will be in its implementation in real systems. The Esprit SysMan and IDSM Projects are implementing Obligation and Authority policies for distributed system management based on a model very similar to ours [Sloman 1993], as the work was derived from the same common base [Moffett 1993b], but we have yet to demonstrate its practicability for safety policies.

ACKNOWLEDGEMENTS

We are grateful for useful comments and discussion from Jon Hall, Damian Marriott, Bret Michael and Morris Sloman, and for the improvements suggested by the anonymous reviewers. This work was supported by UK SERC grant GR/G49531.

REFERENCES

Anderson, J.P. (1972). Computer Security Technology Planning Study. ESD/AFSC Hanscom, AFB Bedford, Mass.

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

Chakravarthy, S. et al. (1989). HiPAC: A Research Project in Active, Time-Constraint Database Management. Xerox Advanced Information Technology, 4 Cambridge Center, Cambridge, MA 02140, USA, Technical Report XAIT-89-02, July 1989.

Dobson, J.E. and J.A. McDermid (1989). A Framework for Expressing Models of Security Policy. Proc. *IEEE Symposium on Security and Privacy*, Oakland, CA, May 1989, IEEE Computer Society Press.

Jonscher, D. (1992). Extending Access Control with Duties. Proc. *IFIP WG 11.3 Sixth Working Conference on Database Security*, Vancouver, August 1992.

Khosla, S. (1988). System Specification: A Deontic Approach. PhD Thesis, Dept of Computing, Imperial College, University of London, UK.

Larrondo-Petrie, M.M., E. Gudes, et al. (1990). Security Policies in Object-Oriented Databases. *Database Security III: Status and Prospects*. Elsevier.

Leitch, R. (1993). Engineering Diagnosis: matching problems to solutions. Proc. *Tooldiag'93*, Toulouse,

Masullo, M.J. and S.B. Calo (1993). Policy Management: An Architecture and Approach. Proc. *IEEE Workshop on Systems Management*, UCLA, California, USA, 14-16 April 1993.

McDermid, J.A. (1994). Support for Safety Cases and Safety Arguments using SAM" *Reliability Engineering and System Safety* **43**: 111.

Michael, J.B., E.H. Sibley, et al. (1991). A Modeling Paradigm for Representing Intentions in Information Systems Policy. Proc. *First Workshop on Information Technologies and Systems*, MIT Sloan School of Management, Cambridge, Mass.

Moffett, J.D. and M.S. Sloman (1993a). Policy Hierarchies for Distributed Systems Management. *IEEE Journal of Selected Areas in Communication* **11**(9): pp. 1404-1414.

Moffett J.D. & M.S. Sloman (1993b). User and Mechanism Views of Distributed System Management, Distributed Systems Engineering Journal, **1**(1), Aug. 1993: pp. 37-47

Moffett, J.D. (1994a). Specification of Management Policies and Discretionary Access Control, Chapter 17 of *Management of Distributed Systems and Computer Networks*., ed. Sloman M.S., Addison-Wesley: pp. 455-479,.

Moffett, J.D. and M.S. Sloman (1994b). Policy Conflict Analysis in Distributed System Management. *Journal of Organizational Computing* **4**(1): pp 1-22.

Moffett, J.D. and J.A. McDermid (1994c). Policies for Safety-Critical Systems: the Challenge of Formalisation. To appear in Proc. *Distributed Systems Operations and Management Workshop (DSOM'94),* Toulouse, France

Nguyen, T.N. (1993). Business-IT Policy: Modeling, Acquisition and Use in Systems Management. Proc. *IEEE/IBM International Conference on Systems Management*, Los Angeles.

Nordvik, J.-P. and M. Wilikens (1993). A real-time expert system for total Loss control: The Formentor system. Proc. *Workshop on Integration in Real-Time Intelligent Control Systems*, Madrid, Spain,

Sibley, E.H., R.L. Wexelblat, et al. (1993). The Role of Policy in Requirements Definition: The Next Challenge for Knowledge-Based Software Engineering. Proc. *IEEE Symposium on Requirements Engineering*, San Diego, 1993.

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

Sloman, M.S. and K.P. Twidle (1994). Domains: A Framework for Structuring Management Policy, Chapter 16 of *Management of Distributed Systems and Computer Networks*., ed. Sloman M.S., Addison-Wesley: pp 433-453.

Stonebraker, M. (1975). Implementation of Integrity Constraints and Views by Query Modification. Proc. *ACM SIGMOD Conference on Management of Data*, San Jose, CA, USA, May 1975.

Storrs, G. (1991). The Policy System. *Knowledge-Based Systems and Legal Applications*. Academic Press. pp 167-182.

Wieringa, R., H. Weigand, et al. (1991). The Inheritance of Dynamic and Deontic Integrity Constraints. *Annals of Mathematics and Artificial Intelligence* **3**, pp 393-428.