Extended version of paper in Proceedings of First International Working Conference on Active Networks (IWAN'99), Berlin, June 1999, , ed. S. Covaci
published by Springer Verlag Lecture Notes in Computer Science

Policy Specification for Programmable Networks

Morris Sloman, Emil Lupu
Department of Computing, Imperial College
London SW7 2BZ, U.K.
Email: m.sloman@doc.ic.ac.uk, e.c.lupu@doc.ic.ac.uk
http://www-dse.doc.ic.ac.uk/ ~mss

19 May 1999

Abstract

There is a need to be able to program network components to adapt to application requirements for quality of service, specialised application dependent routing, to increase efficiency, to support mobility and sophisticated management functionality. There are a number of different approaches to providing programmability all of which are extremely powerful and can potentially damage the network, so there is a need for clear specification of authorisation policies i.e., who is permitted to access programmable network resources or services. Obligation policies are event triggered rules which can perform actions on network components and so provide a high-level means of 'programming' these components. Both authorisation and obligation policies are interpreted so can be enabled, disabled or modified dynamically without shutting down components. This paper describes a notation and object-oriented framework for specifying policies related to programmable networks and grouping them into roles. We show how abstract, high-level policies can be refined into a set of implementable ones. The specification of policies may also be distributed which can lead to conflicts. The types of modality and semantic policy conflicts which need to be detected and resolved are discussed.

Keywords:

Management policy, security policy, authorisation, permission, obligations, roles, policy conflicts, policy refinement, domains.

1 INTRODUCTION

There is a need for networks to become more adaptable to cater for the wide range of user devices ranging from powerful multi-media workstations to hand-held portable devices. A convergence is taking place between telecommunications and computing so networks are increasingly being used to transport voice, video, fax as well as data traffic. Future personal digital assistants will include mobile phones and Web-enabled mobile phones are beginning to appear.

There is a need to reconcile the perspectives of the computing and telecommunication communities in new dynamically programmable network architectures that support fast service creation and resource management through a combination of network aware applications and application aware networks. It is necessary to be able to dynamically program the resources within a network to permit adaptive quality of service management, flexible multicast routing from multiple sources for applications such as video conferencing, intelligent caching and load distribution for Web servers or to perform compression and filtering when traversing low bandwidth wireless links. These types of application specific functions need to be dynamically programmed within the network components in order to support flexible and adaptive networks. The main objective is to speed up the slow evolution of network services by building programmability into the network infrastructure itself (Wetherall, 1998).

There are a number of approaches to supporting Programmable Networks:

Active Networks – in addition to carrying data, the packets traversing the network contain programs which invoke operations supported by switches and routers (Tennenhouse, 1997). Example uses include setting up multicast routing groups or fusion of data from many different sensors into larger messages to traverse the network to the data sink. This is essentially programming at the IP level and is often limited to routing or filtering type operations. However, it still has inherent security risks which can be alleviated by the use of 'safe' languages or executing the programs in a controlled environment such as an associated processor rather than the main processor within a network component.

Mobile Agents – agents containing code and state information traverse multiple nodes within a network in order to perform functions on behalf of users e.g., an email to voice converter which follows a mobile phone user (Bieszczad, 1998). This type of programming is generally associated with hosts or servers connected to the network rather than switches or routers but could also be used to set up specific routing tunnels within a network (de Meer, 1998).

Management Interface – network components provide an interface for performing management operations which facilitates a limited form of programming of components to change their behaviour by invoking these operations (Lazar, 1997). This is really provided for the use of managers of the network service but some operations may be made available to managers of value-added, third-party service providers or even user applications. For example there could be service creation and service operation interfaces to support various virtual network, multicast or multimedia services. IEEE are standardising an Applications Programming Interface for Networks [http://www.ieee-pin.org/].

Management by Delegation – this is a means of downloading management code to be executed within network components to perform functions such as complex diagnostic tests on specific nodes. This is an extension to the Management Interface approach as it supports remote execution of code rather than just remote operation invocation. Code delegation is usually performed by network managers but could be used to load specific filtering or compression code onto an access gateway on behalf of an application or user. The advent of Java has made it easier to implement portable 'elastic agents' into which code can be loaded dynamically.

Interpreted Policy – there has been recent interest in bandwidth management policies which specify who can use network resources and services based on time of day, network utilisation or other

application specific constraints (3Com98). Most of the previous work on policy has been related to management of distributed systems and networks (Sloman 1994a, Magee 1996). Authorisation policies specify what actions a subject is permitted or forbidden to perform on a set of target objects. Obligation policies specify what actions must be performed by a subject on a target. Policies are interpreted by agents within the network, and can be used to modify the behaviour of network components so can be considered a 'constrained' form of programming (Sloman, 1994).

There is no single universal solution to programmability of networks and the various approaches can be used to perform complementary functions, although there is some overlap between them as a particular functionality could be implemented using more than one approach. In addition, these are all very powerful facilities which can easily destroy the normal working of the network so it is necessary to specify authorisation policies to define who can program specific components and what programming operations they can access. The obligation policies are event triggered rules which result in actions being performed. This can be considered a 'constrained' form of programming in that policies can be dynamically modified but can only call predefined actions. The policies can be used to define event triggered calls on a management interface, the event conditions or constraints on loading or executing code in an elastic agent. Thus, the policy-based approach is complementary to other approaches described above.

In the remainder of this paper we will concentrate on how to specify policies for the adaptability and security needed in programmable networks. Section 2 outlines how objects can be grouped in domains in order to apply a common policy while the policy notation and implementation are described in Sections 3 and 4. Section 5 is devoted to the study of the conflicts which may arise in a set of policies and covers both modality and semantic conflicts. Policies can be grouped into roles which specify the rights and duties of managers (human or automated) in the network. The specification and use of roles is covered in Section 6, while Section 7 introduces object-oriented extensions to the role model for instantiating role-instances from common class specifications. Policies for the configuration and management of network devices are not specified in isolation but derived from business objectives and requirements. Section 8 gives further examples of policy specifications and discusses the issues relating to the refinement of policies from an abstract description to implementable rules. Related work is discussed in Section 8 and the conclusions are presented in Section 10.

2 DOMAINS & DIRECTORIES

In large-scale systems it is not practical to specify policies for individual objects and so there is a need to be able to group objects to which a policy applies. For example, a bandwidth management policy may apply to all routers within a particular region or of a particular type. An authorisation policy may specify that all members of a department have access to a particular service. Domains provide a means of grouping objects to which policies apply and can be used to partition the objects in a large system according to geographical boundaries, object type, responsibility and authority or for the convenience of human managers (Sloman 1994a&b). Membership of a domain is explicit and not defined in terms of a predicate on object attributes. A domain does not encapsulate the objects it contains but merely holds references to object interfaces. A domain is thus very similar in concept to a file system directory but may hold references to any type of object, including a person. A domain, which is a member of another domain, is called a **sub-domain** of the parent domain. A sub-domain is not a subset of the parent domain, in that an object included in a sub-domain is not a *direct* member of the parent domain, but is an *indirect* member, c.f., a file in a sub-directory is not a direct member of a parent directory. An object or sub-domain may be a member of multiple parent domains. For example, in Figure 1, the 2 'bean people' and sub-domain E are members of both B and C domains which therefore **overlap**. Details of domains are described in (Sloman, 1994a&b).

Path names are used to identify domains, e.g., domain E can be referred to as /A/B/E or /A/C/E as an object may have different local names with multiple parent domains, where '/' is used as a delimiter for domain path names. Policies normally propagate to members of sub-domains, so a policy applying to domain B will also apply to members of domains D and E. **Domain scope expressions** can be

used to combine domains to form a set of objects for applying a policy, using union, intersection and difference operators, e.g., a scope expression @/A/B + @/A/C - @/A/B/E would apply to members of B plus C but not E, and @/A/B ^ @/A/C applies only to the direct and indirect members of the overlap between B and C. The '@' symbol selects all non-domain objects in nested domains.

An advantage of specifying policy scope in terms of domains is that objects can be added and removed from the domains to which policies apply without having to change the policies. However, objects have to be explicitly included in domains. It is not practical to specify domain membership in terms of a predicate based on object attributes but a policy can select a subset of members of a domain and it's subdomains, to which it applies, by means of a constraint in terms of object attributes (see section 3).

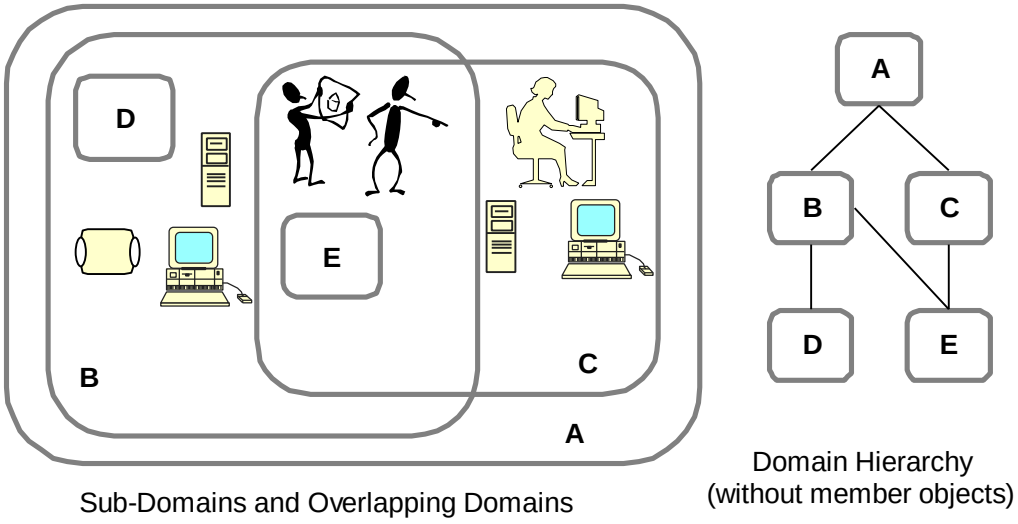

Figure 1 Domains

We have implemented our own domain service but we are now investigating the use of the LDAP (Lightweight Directory Access Protocol) directory service (Whal, 1997) to implement the required functionality. Although LDAP supports the concept of an alias as a reference to an object in another domain, it does not permit objects to be members of multiple directories.

3 POLICY NOTATION

A precise notation is needed for system administrators and (technical) users to specify the network policies related to the applications or services for which they are responsible. This notation is the means of 'programming' the automated agents in network components which interpret policy but can also be used to specify higher level abstract policies or goals which are interpreted by humans or are refined into implementable policies (Marriott 1996a&b, 1997). Another reason to have a precise notation is that policies may be specified by multiple distributed administrators so conflicts between policies can arise. Our notation can be analysed by tools to detect and, in some cases, resolve conflicts.

The policies are interpreted rather than compiled into the code of agents, so can be changed dynamically. Implementable policies are directly interpreted by automated manager and access control agents, which are (potentially) distributed, so we do not use logical deduction in order to analyse the state of the system.

Authorisation policies define what activities a subject can perform on a set of target objects and are essentially access control policies to protect resources from unauthorised access. Constraints can be

specified to limit the applicability of both authorisation and obligation policies based on time or values of the attributes of the objects to which the policy refers.

x1 A+ @/Network_Admin {PolicyObjType: load(); remove(); enable (); disable () } @/Nregion/switches

Members of the Network_Admin domain are authorised to load, remove, enable or disable policies in the Nregion switches domain. The ';' is used to separate the permitted actions.

x2 A- n: @/test-engineers { performance_test() } @/routers when n.status = trainee

Trainee test engineers are forbidden to perform performance tests on routers. Note the use of a constraint based on subject state information

x3 A+ @/Agroup + @/Bgroup {VideoConf (BW=2, Priority=3)} USAStaff – NYgroup
when (16:00 < time < 18:00)

Members of the Agroup plus Bgroup can set up video conference with a bandwidth of 2 Mb/s and priority of 3, with USA staff except the New York group, between 16:00 and 18:00. Note the use of a time based constraint.

Obligation policies define what activities a manager or agent must or must not perform on a set of target objects. Positive obligation policies are triggered by events.

x4 O+ on video_request(bw, source) @/USGateway { router:bwreserve (bw); log(bw, source)}
@/routers/US

This positive obligation policy is triggered by an external event signalling that a video channel has been requested. The object in the USGateway domain first does a bwreserve operation on all objects of type router in the /router/US domain and then logs the request (assume to an internal log file) i.e., operations specified in a policy can be on external objects or internal operations in the agent. The ';' is used to separate a *sequence* of actions in a positive obligation policy.

x5 O+ at 01:00 @/archiver { backup (); reset () } @/logfile

This positive obligation policy is triggered by an internal event, every night at 1 am, for the archiver to backup and reset the logfile.

x6 O- n:@/test-engineers { DiscloseTestResults() } @/analysts + @/developers
when n.testing_sequence == in-progress

This negative obligation policy specifies that test engineers must not disclose test results to analysts or developers when the testing sequence being performed by that subject is still in progress, i.e., a constraint based on the state of subjects.

The general format of a policy is given below with optional attributes within brackets (the braces and semicolon are the main syntactic separators). Some attributes of a policy such as trigger, subject, action, target or constraint may be comments (e.g. */* this is a comment */*), in which case the policy is considered high-level and not able to be directly interpreted.

identifier mode [trigger] subject '{' action '}' target [constraint] [exception] [parent] [child] [xref] ';'

The **identifier** is a label used to refer to the policy. The **mode** of the policy distinguishes between positive obligations (**O+**), negative obligations (**O-**), positive authorisations (**A+**) and negative authorisations (**A-**).

The **trigger** only applies to positive obligation policies. It can specify an internal timer event using an **at** clause, as in x5 above, or an **every** clause for repetitive events. An external event is defined using an **on** clause, as in x4 above, where the video_request event passes parameters bw & source to the agent. These events are detected by a monitoring service. The policy notation only specifies simple events as a generalised monitoring service can be used to combine complex event sequences to generate simple events (Mansouri-Samani, 1996).

The **subject** of a policy, defined in terms of a domain scope expression, specifies the human or automated managers and agents to which the policies apply and which interpret obligation policies. The **target** of a policy, also defined in terms of a domain scope expression, specifies the objects on which actions are to be performed. Security agents at a target's node interpret authorisation policies and manager agents in the subject domain interpret obligation policies.

The **actions** specify what must be performed for obligations and what is permitted for authorisations. It consists of method invocations or a comment and may list different methods for different object types. Multiple actions in an authorisation policy indicate the set of actions or operations which are permitted or forbidden. Multiple actions in a positive obligation policy imply that they are performed sequentially after the policy is triggered.

The **constraint,** defined by the **when** clause, limits the applicability of a policy, e.g. to a particular time period as in policy x3 above, or making it valid after a particular date (**when** time > 1/June/1999). In addition, the constraint could be based on attribute values of the subject such (as in policy x2 above) or target objects. In x2, the label n, prepended to the subject, is referenced in the constraint to indicate a subject attribute. Constraints must be evaluated every time an obligation policy is triggered or authorisation policy is checked to see whether the policy still applies as attribute values may change.

An action within an obligation policy may result in an operation on a remote target object. This could fail due to remote system or network failure so an **exception** mechanism is provided for positive obligations to permit the specification of alternative actions to cater for failures which may arise in any distributed system.

High-level abstract policies can be refined into implementable policies. In order to record this hierarchy, policies automatically contain **references** to their parent and children policies. In addition, a cross-reference (xref) from one policy to another can be inserted manually, e.g., so that an obligation policy can indicate the authorisation policies granting permission for its activities (see section 8).

4 POLICY IMPLEMENTATION ISSUES

The policy service provides tool support for defining policies and disseminating polices to the relevant agents that will interpret them. Policies are implemented as objects which can be members of domains so that authorisation policies can be used to control which administrators are permitted to specify or modify policies stored in the policy service.

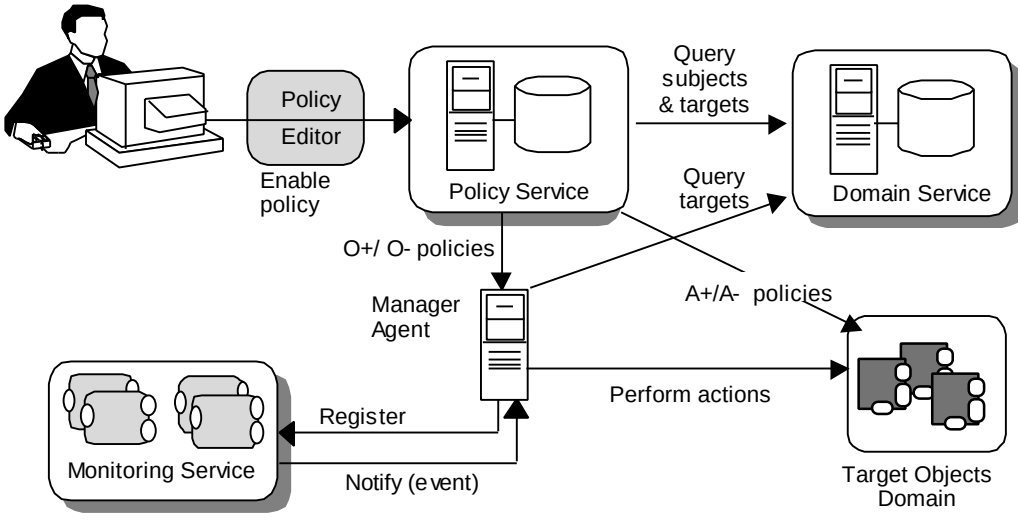

Figure 2 Policy Enforcement

An overview of the approach to policy enforcement is given in Figure 2. An administrator creates and modifies policies using a policy editor. He checks for conflicts, and if necessary modifies policies to remove the conflicts (see section 5). Authorisation policies are then disseminated to target security agents as specified by the target domains and obligation policies to automated manager agents as specified by the subject domains. Policies may be subsequently enabled, disabled or removed from the agents. Manager agents register with the monitoring service to receive relevant events generated from the managed objects. On receiving an event which triggers one or more obligation policies, the agent queries the domain service to determine target objects and performs the policy actions, provided no obligation policies restrain the action.

Figure 3 shows a policy agent which interprets obligation policies. It is application specific in that there can be agents for quality of service management which are different from those used for security management, for example. Each class of agent has predefined management functions which are accessible from the policies. These functions may result in operations on remote target objects or can be internal to the agent. The functionality of an agent could be dynamically modified using Management by Delegation techniques to load new code, but this has not been implemented in our prototype. More details on the syntax, and implementation issues of the policy service can be found in (Marriott 1996a, 1996b and 1997).

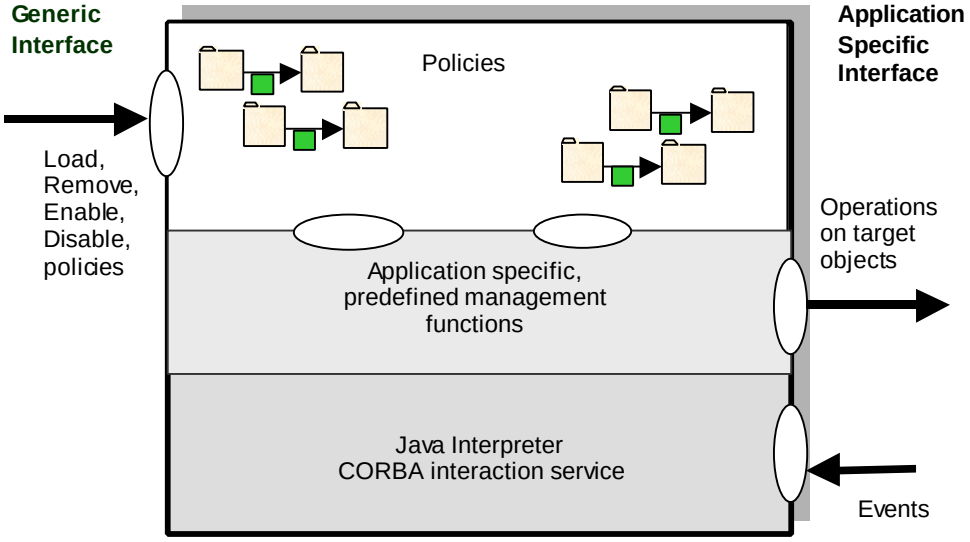

Figure 3 Obligation Policy Agent

5 POLICY CONFLICTS

In any large inter-organisational distributed network, policies are specified by multiple managers, possibly within different organisations. Objects can be members of multiple domains so multiple policies will typically apply to an object. It is quite possible that conflicts will arise between multiple policies. There are two types of conflicts which we will consider – modality and semantic conflicts.

5.1 Modality Conflicts

These are inconsistencies in the policy specification which may arise when two or more policies with modalities of opposite sign refer to the same subjects, actions and targets. This occurs when there is a triple overlap between the sets of subjects, targets and actions as shown in Figure 4, and so can be determined by syntactic analysis of polices. Figure 4 shows 2 policies – P1 with positive modality, subjects in s1, targets in t1 and actions a1; P2 with negative modality, subject in s2, targets in t2 and actions a2. The overlaps are indicated by sc, ac, and tc.

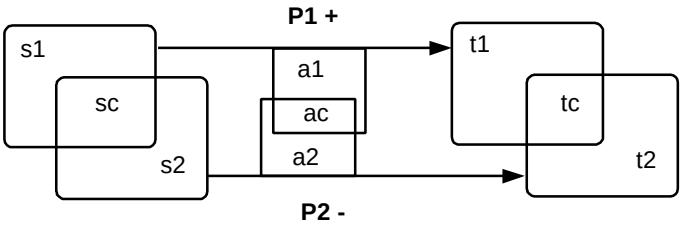

Figure 4 Overlapping Subjects, Targets and Actions

There are three types of modality conflicts:

- **O+/O-** the subjects are both required and required not to perform the same actions on the target objects.
- **A+/A-** the subjects are both authorized and forbidden to perform the actions on the target objects.
- **O+/A-** the subjects are required but forbidden to perform the actions on the target objects.

Note that O-/A+ is not a conflict, but may occur when subjects must refrain from performing certain actions as specified by a negative obligation, even though they are permitted to perform the actions, as in policy X6 in section 3.

An example conflict occurs in the following policies

X7 **A-** @/VisitingAgents {create(); delete (); write ()} @/local/files

Visiting mobile agents are forbidden from creating, deleting or writing to any local files in the host node.

X8 **A+** n:@/VisitingAgents { write () } @/local/files/netman
when authenticate (n.owner=netman)

A visiting agent, whose authenticated owner is netman, is permitted to write to any local files in the netman subdomain.

Policy X7 permits an action which X8 forbids, so a modality conflict occurs. It is possible to resolve these conflicts automatically by assigning a priority to individual policies, but meaningful priorities are notoriously difficult for users to assign and may result in arbitrary priorities which do not really relate to the importance of the policies. Inconsistent priorities could easily arise in a distributed system with several people responsible for specifying policies and assigning priorities. Our approach has been to permit more specific policies to have precedence – a policy applying to a sub-domain overrides more general policies applying to an ancestor domain. We refine X8 into 2 policies, X9 and X 10 below, to permit automatic conflict resolution. In X9 the positive authorisation applies to a subdomain of both subject and target so is more specific than X7 and therefore would have precedence over X7. Policy X10 would also be needed to make sure only mobile agents with authenticated owner of netman can enter the Owners subdomain.

X9 **A+** n:@/VisitingAgents/Owners { write () } @/local/files/netman

X10 **A+** n:@/ VisitingAgents { enter () } @/VisitingAgents/Owners
when authenticate (n.owner=netman)

We have implemented tools for analysing a domain of policies to indicate conflicts for an administrator to resolve. The specifity precedence can enabled or disabled. We are investigating techniques for specifying other forms of precedence – in some situations negative authorisation policies should have precedence over positive ones, more recent policies over older ones or perhaps policies applying to short time-scales over longer (background) ones.

5.2 Semantic Conflicts and Metapolicies

Modality conflicts can be detected purely by syntactic analysis of the policies. Application-specific conflicts arise from the semantics of the policy. For example, policies must not allocate more than

20% of the available bandwidth to any single request; policies related to differentiated services which define to which queues specific types of packets should be allocated, must not result in 2 different queues to which the packet should be allocated or two or more policies triggered by different events which occur at approximately the same time may have conflicting actions such as to increase and decrease a message buffer size. These conflicts for resources or conflicts of action are application specific and cannot be detected automatically without a specification of what is a conflict i.e., the conflicts are specified in terms of constraints on attribute values of *permitted* policies. We call these constraints **metapolicies** as they are policies about which policies can coexist in the system or what are permitted attribute values for a valid policy. For example, a semantic conflict may arise if there are two policies which increase and decrease bandwidth allocation when the same event occurs. Although this resembles a syntactic conflict rather than a semantic one, the conflict arises only because the actions are respectively to 'increase' and 'decrease' the bandwidth allocation.

Meta-Policies can be expressed in a logical notation such as Prolog. The evaluation of a Meta-Policy, written as a Prolog predicate, must not only detect the presence of conflicts but also determine the policies in conflict. Thus the solutions of the predicate evaluation must be the conflicting policies. For example, two positive obligation policies (applying to the same subject) must not assign the same type of traffic to different queues. This can be written as a logical predicate in the form:

$\forall$P1, P2 $\in$ @policies/traffic-queuing

```
conflict(P1,P2) :- equals(mode(P1), mode(P2), 'O+'),
                   equals(trigger(P1), trigger(P2)),
                   equals(action(P1), action(P2), 'assign(traffic-type)'),
                   notequals(target(P1), target(P2)).
```

All the solutions P1, P2 satisfying this predicate satisfy the conflict condition and are thus in conflict. The policies must then be revised according to the desired traffic allocation strategy. Metapolicies must be checked at specification time or when the policies are loaded into the subject. In the example above, the metapolicy refers only to policies within the same router and hence must be checked when policies are loaded into the subject. It can also be checked at specification time if a check for subject overlap is included in the metapolicy specification (i.e., by adding intersect(subject(P1), subject(P2), X) to the conjunction). However, this widens the scope of the conflict check to all traffic-queuing policies within the managed network and the computational cost may be prohibitive.

Often, the subject and targets sets of a policy are only determined at run-time when the domain scope expressions are evaluated. Furthermore, parameter values for policy actions may depend on attribute values of the triggering events specified in a constraint. Hence, conflict checking at specification or at load time may detect only potential rather than actual conflicts. It is nevertheless essential to perform off-line static analysis and propagate notifications of changes (e.g., to domain membership) in order to alleviate the cost of run-time conflict detection.

6 ROLES

Organisational structure is often specified in terms of **organisational positions** such as regional, site or departmental network manager, service administrator, service operator, company vice-president. Specifying organisational policies for people in terms of role-positions rather than persons, permits the assignment of a new person to the position without re-specifying the policies referring to the duties and authorisations of that position. The tasks and responsibilities corresponding to the position are grouped into a role associated with the position (which is essentially a static concept in the organisation). The position could correspond to a manager or a user of a network or services. A **role** is thus the position, the set of authorisation policies defining the rights for that position and the set of obligation policies defining the duties of that position. These definitions correspond to the concepts of classic Role Theory which postulates that individuals occupy positions inside an organisation and associated with the position are a set of activities (including the required interactions) that constitute the role of that position (Biddle, 1979).

Organisational positions can be represented as domains and we consider a **role** to be the set of authorisation and obligation policies (the arrows in Figure 5) with the **Position Domain** as subject. A person or automated agent can then be assigned to or removed from the position domain without changing the policies as explained in (Lupu, 1997a).

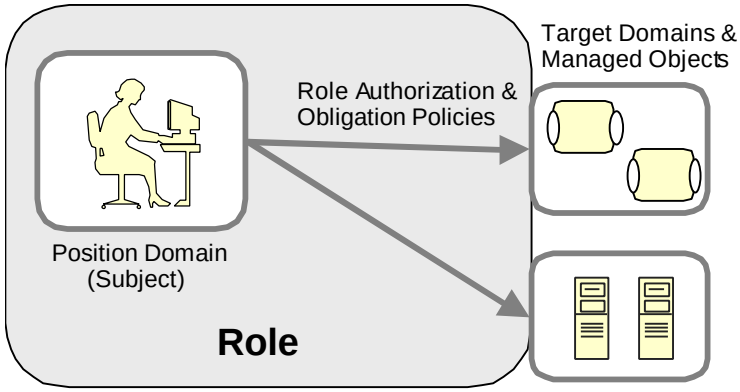

Figure 5 Management Roles

Although the concept of role was originally defined to apply to people, it can also be used to group the authorisation and obligation policies which apply to a particular type of network component as a subject e.g., an edge-router that interconnects the local network to the service provider or a core-router which provides a backbone service. It is possible that similar hardware and software is used for both core and edge routers and so assigning a particular router to a role will define the particular set of policies which are loaded onto that router. Another example is a visiting agent role which is assigned to a mobile agent when it is received at a network node. This could specify what resources it can access and what actions it must perform on arrival and departure.

There are additional extensions to the concepts of roles described in (Lupu, 1997b & 1998). These define inter-role relationships in terms of interaction protocols and concurrency constraints on the ordering of obligation actions, however they will not be discussed further in this paper.

7 OBJECT ORIENTED APPROACH

An organisation may contain large numbers of roles with few differences between them e.g. there can be thousands of instances of students, hundreds of instances of lecturers and 20-50 departmental network managers in a university. We introduce classes and templates in order to reduce the number and complexity of the specifications. For example a departmental network manager role class can specify common rights and duties and used to create the manager-instance roles for each department. Similarly a lecturer and or student role-class defines the types of services they can access. Each instance may then be customised for any particular task relating to a specific department and a specific person assigned to each role. Inheritance is used for specialising role classes from more generalised classes. For example, an edge-router role could be a specialisation of the core-router role with some modified or additional policies. In this section, we define the role object model and examine its uses. The definition of role classes is based upon policy templates (which are specifications of obligations and authorisations independent of subject, target or both).

7.1 Policy Templates

Policy templates use variables to represent subjects and/or targets i.e., they specify the policy actions and constraints which can be reused for different subjects and targets according to the application.

The instantiation of a policy object from a policy template is done by specifying the subjects and targets. The following two templates can be used to set up different quality video conference sessions depending on the time of day.

A+ @/S {VideoConf (BW=2, Priority=1)} T **when** (09:00 < time < 18:00)

A+ @/S {VideoConf (BW=4, Priority=4)} T **when** (07:00 < time < 09:00)

The following policy instances can be created, by assigning values to S and T, authorising a DoC lecturer to set up low quality video conferences to UK sites during normal office hours and to set up high quality conferences to USA sites only early in the morning.

A+ @/IC/DOC/lecturers {VideoConf (BW=2, Priority=1)} @destinations/UK
when (09:00 < time < 18:00)

A+ @/IC/DOC/lecturers {VideoConf (BW=4, Priority=4)} T @destinations/USA
when (07:00 < time < 09:00)

A policy template may not inherit from another policy template since the components of a policy (actions, condition, trigger, etc.) are closely related and cannot be combined by inheritance. The policy will maintain a reference to the template from which it has been created in addition to the references maintained to the policies it has been refined from. When a policy template is instantiated from a role class, only the target has to be specified since the subject is determined by the position domain (see Section 7.2). Policy templates can also be used to instantiate policies independently of roles.

7.2 Defining Role Classes

A role class groups authorisation and obligation policy template specifications. When a role instance is created, a position domain for the role is then created so a person or component cannot be assigned to a role *class*, only to a role *instance*. The role class contains only policy templates, not instances. Assume a scenario where each edge router has a direct connection to a specific core router in the backbone network. The edge router is permitted to update the multicast set of destinations for the core router, related to a specific application. In addition every hour the edge router sends its error-readings to a central monitor node. The policy template authorising reading, updating or clearing multicast lists for an application A would be:

pt_1 **A+** D {Update (A, DestList); Read(A, DestList); Clear (A)} T

pt_2 **O+** every (60 min) D {sendErrors()} @monitors/errors

When the role is instantiated for Department DOC a DOC_EdgeRouter domain will be created and assigned to the variable D for both p1_1 and pt_2. A specific core router domain (e.g. CR5) will be assigned to the variable T in pt_1 but the target in pt_2 has been pre-specified as it is the same for all policy instances.

An alternative to specifying the pt_2 policy template as part of the edge-router role, would be to define a global policy instance:

G3 O+ **every** (60 min) @/Departments/EdgeRouters {sendErrors()} @monitors/errors

The disadvantage of this approach is that the every EdgeRouter instance, when it is installed, must be included into the Department/EdgeRouter domain which is rather difficult to implement and to check. With the role approach, when the role is created and a router assigned to the DOC_EdgeRouter domain, all obligation policies will be loaded into the router and all authorisation policies will be sent to the relevant targets (CR5 for the policy instance created from pt_1).

7.3 Inheritance

Single and multiple inheritance can be defined between role classes in order to implement specialisation and re-use of the specifications. Figure 6 shows the Network manager inheriting from both network operations and manager role classes, while the network operator inherits only from the network operations class. Currently we only support class-based inheritance although some work in the Role Based Access Control (RBAC) community advocates the need for inheritance of role instances (Sandhu, 1996). This optimises the re-use of role attributes (permissions in RBAC) in

situations where a new role has similar duties and rights over the *same target objects* as the one it is derived from. However, this approach has been criticised since it uses inheritance to model different types of relationships between roles without distinguishing between them (Moffett, 1998). Furthermore, it is not clear that supporting too many different forms of inheritance is a good thing as it makes the model more difficult to understand, and the support tools more complicated.

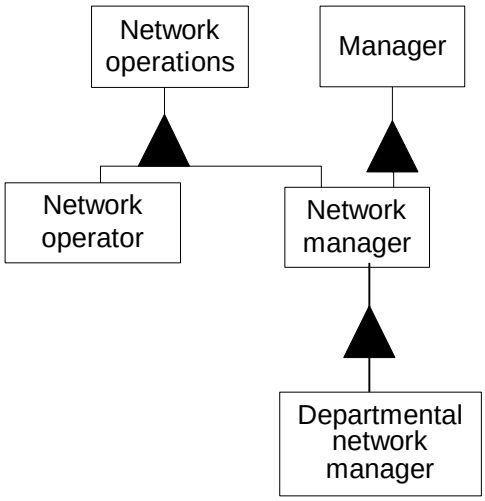

Figure 6 Role Class Inheritance

8 POLICY REFINEMENT

High-level abstract policies are often specified as part of the business process and express requirements from the communication network. These requirements are specified as management goals which cannot be directly interpreted by automated components and hence, must be refined into functional policy specifications or be implemented manually by human managers. We express abstract policies in the same notation as implementable policies, however the policy attributes (subjects, actions, etc.) may be written in natural language. For example, a high-level policy may be written as:

T1 O+ @/NetworkManagers { /* maintain adequate video conference set up */ }
@/users/groupA when 14:00 < time < 15:00

Network managers must maintain an adequate video conference set up for groupA users between 14:00 and 15:00.

In order to achieve this goal it is necessary to refine the policy into access control policies which ensure that group A users can set up video conference links, bandwidth management policies and further administrative policies to enable or disable any special policies which might apply during these hours. For example:

Administrative policies

T2 O+ at 13:55 @/NetworkManagers { enable() }
@/policies/specialBandwidthControl + @/policies/QoSmonitoring

Network managers must enable special bandwidth control and QoS monitoring policies at 13:55.

T3 O+ at 15:00 @/NetworkManagers { disable() }
@/policies/specialBandwidthControl + @/policies/QoSmonitoring

Network managers must disable the special bandwidth control and QoS monitoring policies after 15:00.

Authorisation policies

T4 A+ @/Agroup {VideoConf (BW=2, Priority=3)} @/USAStaff when (14:00 < time < 15:00)

Group A users must be able to set up the video connections (similar to policy x3).

T5 A+ @/NetworkManagers { enable(); disable() }
@/policies/specialBandwidthControl + @/policies/QoSmonitoring

Network are authorised to enable and disable special bandwidth control and QoS monitoring policies.

Bandwidth Control

T6 O+ @edgeRouters { /* optimise reserved bandwidth*/ } @/channels

Edge routers must maintain adequate reservations for channels which they control. This policy is also an abstract policy which needs to be refined; for example in:

T7 O+ on request(bw,channelId) @/edgeRouter {reduceReservation(bw) }
@/channels/channelId
when bw < getReservation(channelId)

Edge routers should decrease the bandwidth reservation when the request is for less than the amount currently reserved.

T8 O+ on request(bw, channelId) @edgeRouter { increaseReservation(min(bw, x)) }
@channels/channelId when bw > getReservation(channelId)

Edge routers should increase bandwidth when the request is for more than the amount currently reserved. However, the amount reserved should not exceed x (i.e. a fixed value).

The refinement of an abstract policies into implementable policy specifications must be done by the human managers. However, as shown by the example above there are no obvious restrictions that can be imposed. The refinement of a positive obligation policy will require specifying some of the authorisation policies giving the subjects the necessary access rights to perform their tasks while the other authorisation policies may have been already specified by the security manager. Similarly, the refinement of an authorisation policy may include obligation policies defining the measures and counter-measures to be taken in case of security violations.

The refinement of a policy does not preserve the policy modality or necessarily apply to the same the subjects or targets. For example, while network managers are responsible for ensuring that the adequate quality of service is provided (policy T1), the edge routers are responsible for performing the bandwidth reservations (T6-T8).

We currently maintain pointers from an abstract to dependent policies, derived from it, (omitted from the above examples for clarity) but we do not have tools to support the refinement process. We would like to be able to verify that the set of refinements fully implements the abstract policy but this is a challenging research problem. Furthermore, semantic conflicts between policies should be detected as high as possible in the refinement hierarchy. We are investigating the use of requirements engineering tools and techniques for refinement and analysis of policies.

The policies above have been specified in terms of domains. However, the policies could also be included in roles e.g., Network Manager, Edge Router, to which users or devices are assigned.

9 RELATED WORK

There are a number of groups working on policies for management of networks and distributed systems (Magee 1996, Weiss 95, Koch 96). Some of this has been based on our early proposals for policy notation. Another approach is to define policy using the full power of a general purpose scripting or interpreted language (eg TCL or Java) and load this into Network components. (Bos 1999) takes this approach to specify application policies for resource management for netlets, which are small virtual networks within a larger virtual network.

There is considerable interest in the internet community in using policies for bandwidth management. They assume policies are objects stored in a directory service [3Com 98]. A policy client (e.g. a router) makes policy requests on a server which retrieves policy objects from a directory server, interprets the policy and responds with policy decisions to the client. The client enforces the policy by, for example, permitting/forbidding a request or allocating packets from a connection to a particular queue.

The IETF are defining a policy framework that can be used for classifying packet flows as well as specifying authorisation for network resources and services (Strassner, 1998 a, b & c). They do not explicitly differentiate authorisation and obligation policies. A simple policy rule defines a set of policy *actions* which are performed when a set of policy *conditions* becomes true. These conditions correspond to a combination of our events and constraints for obligation policies. A policy may itself be an aggregation of policy rules. This has some similarity to our idea of policy refinement as indicated in Section 8. They also have realised policy conflicts can occur, but have not distinguished between modality and semantic conflicts nor do they say how these conflicts will be detected. Directories are used for storing policies but not for grouping subjects and targets. They have a dynamic group which can be specified by enumeration or by characterisation i.e., a predicate on object attributes. We can achieve this by means of a constraint on policies *within* the scope of a domain expression which is a defined set. Defining a group in terms of an arbitrary predicate can be impractical. For example the group of all Pentium II workstations with memory > 128 Mbytes would require checking millions of workstations on the internet to determine if they are members of the group which would not be feasible. They have the concept of a role which is defined as a label indicating a function that an interface or device in the network serves. Roles enable administrators to group the interfaces of multiple devices into common groups for applying a common policy. This is similar to our domains although it is not clear how it will be implemented. There is a restriction that their role can be associated with a single policy (which can be as complex as necessary). We think this is very restrictive and unnecessary. In the IETF approach a policy *enforcement* point queries a *decision* point to find out which policies apply. Our notation, with explicit subjects and targets permits us to propagate policies to where they are required so we combine decision and enforcement at subjects for obligation policies and targets for authorisation policies. Our policy service disseminates policies to the relevant distributed agents.

Sandhu (1996) presents constraints that are similar to our Meta-policies, but the notation used and the enforcement of the constraints are not described. Minsky's "law governed systems" (Minsky, 1997) can also specify permissions and prohibition as a set of rules which are similar to our positive and negative authorizations. Conflicts are avoided by defining a Meta-level rule which specifies whether a permission or prohibition take precedence and override the other. Minsky (1995) also introduces the use of obligations in order to ensure integrity in access control systems. This is similar to the use of our obligation policies in the refinement of a security policy in order to specify pro-active or coercive measures for preventing security violations.

10 CONCLUSIONS

We have shown that our policy and role approach, although conceived for management of distributed systems is also very useful for programmable networks. A clear specification of authorisation policy is essential, whatever implementation techniques are being used. The obligation policies can be used to 'program' the network components or combined with other programming approaches to define the events and constraints for performing actions.

In any large scale system, conflicts between policies will occur. We distinguish between modality and semantic conflicts and indicate an approach for specifying what is a semantic conflict as a metapolicy. Where possible, conflicts should be detected at specification or load time (c.f. type conflicts detected by a compiler), although some conflicts can only be detected at run-time.

We have also shown the use of roles for specifying policies for network managers, service users and for network components. We have a prototype role framework toolkit which can be used to specify roles and policies. It also performs static analysis for conflicts. We are currently working on extending this to run-time analysis and are investigating the applicability of requirements engineering approaches for refining high level goals into detailed specifications to policy refinement. They also have more sophisticated consistency analysis tools which may be applicable.

11 ACKNOWLEDGEMENTS

We gratefully acknowledge financial support from the Fujitsu Laboratories and British Telecom and acknowledge the contribution of our colleagues to the concepts described in this paper – in particular Nicholas Yialelis and Damian Marriott.

12 REFERENCES

3COM, (1998) Directory Enabled Networking and 3COM's Framework for Policy Powered Networking from http://www.3com.com/

Bieszczad A, Pagurek B, White T. (1998) Mobile Agents for Network Management, *IEEE Communications Surveys*, Vol. 1 No. 1, 1998 www.comsoc.org/pubs/surveys.

Bos H. (1999) Application Specific Policies: Beyond the Domain Boundaries, IFIP/IEEE Integrated Management Symposium (IM'99), Boston, May 1999

de Meer, H., Richter, J.-P., Puliafito, A. and Tomarchio, O. (1998). Tunnel Agents for Enhanced Internet QoS. *IEEE Concurrency* 6(2): 30-39, April-June 1998.

Goldszmidt, G. and Y. Yemini (1993) Evaluating Management Decisions via Delegation, in Integrated Network Management III, eds. H.-G. Hegering, Y. Yemini, Elsevier Science Publisher, pp 247-257.

Koch, T. et al. (1996). Policy Definition Language for Automated Management of Distributed System. Proceedings of the 2nd IEEE International Workshop on Systems Management, Toronto (Canada), June 1996, pp. 55-64.

Lazar, A. (1997) Programming Telecommunication Networks, *IEEE Network*, Sep./Oct. 1997, pp 8-18.

Lupu, E. and M. Sloman (1997a) Towards a Role-based Framework for Distributed Systems Management. *Journal of Network and Systems Management*, **5**(1):5-30, March 1997, Plenum Press.

Lupu E. and M. Sloman (1997b) A Policy-based Role Object Model, Proceedings of the 1st IEEE Enterprise Distributed Object Computing Workshop (EDOC'97*)*, Gold Coast, Australia, Oct.97,pp. 36-47.

Lupu, E. (1998) A Role-Based Framework for Distributed Systems Management. Ph.D. Dissertation, Imperial College, Department of Computing, London, U.K, July 1998.

Magee J. and J. Moffett eds. (1996) Special Issue of *IEE/BCS/IOP Distributed Systems Engineering Journal* on Services for Managing Distributed Systems, **3**(2), June 1996.

Mansouri-Samani M. and M. Sloman (1997). GEM: A Generalised Event Monitoring Language for Distributed Systems, *IEE/BCS/IOP Distributed Systems Engineering*, **4**(2):96-108, June 1997.

Marriott, D. and M. Sloman (1996a). Management Policy Service for Distributed Systems. Proceedings of the 3rd IEEE International Workshop on Services in Distributed and Networked Environments (SDNE 96), Macau, pp. 2–9.

Marriott, D. and M. Sloman (1996b) Implementation of a Management Agent for Interpreting Obligation Policy. Proceedings of the IEEE/IFIP Distributed Systems Operations and Management Workshop (DSOM' 96), L'Aquila (Italy), Oct. 1996.

Marriott, D. (1997) Management Policy for Distributed Systems, Ph.D. Dissertation, Imperial College, Department of Computing, London, UK, July 1997.

Moffett, J. D. (1998) Control Principles and Access Right Inheritance through Role Hierarchies. Proceedings of the 3rd ACM Workshop on Role Based Access Control, Fairfax Virginia, Oct. 1998.

Minsky, N. H. and A. D. Lockman (1985). Ensuring Integrity by Adding Obligations to Privileges. Proceedings of the 8th International Conference on Software Engineering, London (U.K.), August 1985, pp. 92-102.

Minsky, N. H. et al. (1996) Building Reconfiguration Primitives into the Law of a System. Proceedings of the 3rd IEEE International Conference on Configurable Distributed Systems (ICCDS 96), Annapolis (Maryland), pp. 89–97.

Minsky, N.H. and Pal, P. (1997) Law-Governed Regularities in Object Systems – Part 2: A Concrete Implementation. *Theory and Practice of Object Systems (TAPOS)*, **3**(2), John Wiley.

Sandhu, R. S. et al. (1996) Role-Based Access Control Models. *IEEE Computer*, **29**(2):38–47.

Sloman, M. (1994a). Policy Driven Management for Distributed Systems. Plenum Press *Journal of Network and Systems Management*, 2 (4):333–360, Plenum Press.

Sloman, M. and Twidle, K. (1994b). Domains: A Framework for Structuring Management Policy. In *Network and Distributed Systems Management.* Sloman M. ed., Addison Wesley, pp. 433–453.

Strassmer J. and E. Elleson (1998a) Terminology for Describing Network Policy and Services, IETF draft work in progress, Aug. 1998. Available from http://www.ietf.org

Strassmer J. and E. Elleson (1998b) Policy Framework Core Information Model, IETF draft work in progress, Nov. 1998, Available from http://www.ietf.org

Strassmer J. and S. Schleimer (1998c) Policy Framework Definition Language, IETF draft work in progress, Nov. 1998, Available from http://www.ietf.org

Tennenhouse D, Smith J, Sincoskie D, Wetherall D, Minden G, A survey of Active Network Research, IEEE Communications Magazine, 35:1 Jan 1997, pp 80-86.

Yialelis, N. and M. Sloman (1996). A Security Framework Supporting Domain-Based Access Control in Distributed Systems, IEEE ISOC Symposium on Network and Distributed Systems Security, San Diego, pp. 26-34, Feb. 1996.

Whal, M., T. Howes and S. Kille (1997) Lightweight Directory Access Protocol (v3), IETF RFC 2251, Dec. 1997. Available from http://www.ietf.org

Wetherall D. Legedza U., Guttag J. (1998) Introducing New Internet Services: Why and How, *IEEE Network*, Special Issue on Active and Programmable Networks, July 1998.

Wies R. (1995). Policies in Integrated Network and Systems Management: Methodologies for the Definition, Transformation and Application of Management Policies. Ph.D. Dissertation, Fakultat fur Mathematik der Ludwig-Maximilians-Universitat, Munchen, Germany, 1995.

Papers by the authors and the Imperial College PhD dissertations can be obtained from http://www-dse.doc.ic.ac.uk/~mss/MSSPubs.html