First International Working Conference on Active Networks, Berlin, Germany, June-July 1999, Springer-Verlag, LNCS.

Policy Specification for Programmable Networks

Morris Sloman, Emil Lupu

Department of Computing, Imperial College, London SW7 2BZ, U.K.
{m.sloman, e.c.lupu}@doc.ic.ac.uk

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 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 and discuss the types of conflicts which can occur between policies.

1 Introduction

Networks have 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 telecommunication and computing 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 [1].

There are a number of approaches to supporting Programmable Networks:

Active Networks – the packets traversing the network contain normal data plus programs which may invoke switch and router operations [2]. 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. It 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 [3]. 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 [4].

Management Interface – network components provide a management interface which facilitates a limited form of programming of components by invoking operations to change their behaviour [5]. This is really provided for the use of network managers 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 – is a means of downloading management code to be executed within network components to perform functions such as complex diagnostic tests on specific nodes [6]. 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 application specific constraints [7]. Most of the previous work on policy has been related to management of distributed systems and networks [8],[9]. 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 can be used to modify the behaviour of network components so can be considered a 'constrained' form of programming [8].

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. Policies can be used to define the event conditions and constraints for invocations on a management interface, or loading or executing code in an elastic agent. Thus, policies are complementary to the other approaches described above.

This paper focuses on the specification of 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. Sections 3 and 4 discuss the policy notation and implementation, followed by some of the conflict detection and resolution issues. Section 6 introduces roles as a means of grouping policies which specify the rights and duties of managers. Policies for the configuration and management of network devices are not specified in isolation but derived from business objectives and requirements, so section 7 addresses the refinement of policies from an abstract description to implementable rules. Related work and conclusions are presented in sections 8 and 9.

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 [8], [10]. A domain does not encapsulate the objects it contains but merely holds references to object interfaces. It 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. Object and sub-domains may be a member of multiple parent domains and may have different local names in each one of them. For example, in Fig. 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 [8], [10].

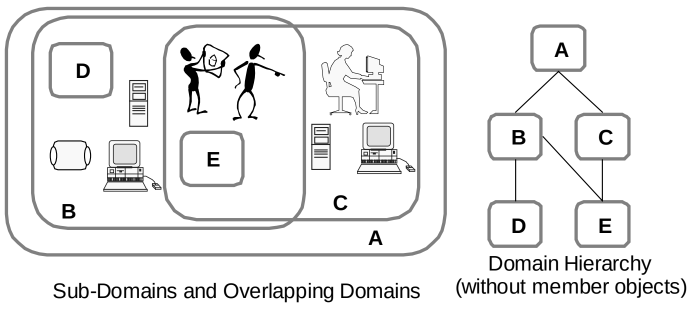

Fig. 1 Domains

Path names are used to identify domains, e.g., domain E can be referred to as /A/B/E or /A/C/E, 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, to which it applies, by means of a constraint in terms of object attributes (see section 3).

We have implemented our own domain service but we are redoing this for an LDAP (Lightweight Directory Access Protocol) directory service [11]. However, 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 [12], [13], [14]. 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. 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+ @/NetworkAdmin {PolicyObjType: load(); remove(); enable (); disable ()}
       @/Nregion/switches
```

Members of the NetworkAdmin domain are authorised to load, remove, enable or disable policies in Nregion/switches. The ';' separates 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 Agroup plus Bgroup can set up a video conference (bandwidth = 2 Mb/s, priority = 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 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 /routers/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- 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. 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 and 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 [16].

The *subject* of a policy, defined in terms of a domain scope expression, specifies the human or automated managers to which the policies apply. 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. An authorisation policy indicates the set of operations which are permitted or forbidden while the multiple actions in a positive obligation policy 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.

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 7).

4 Policy Implementation Issues

The policy service provides tool support for defining and disseminating polices to the 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.

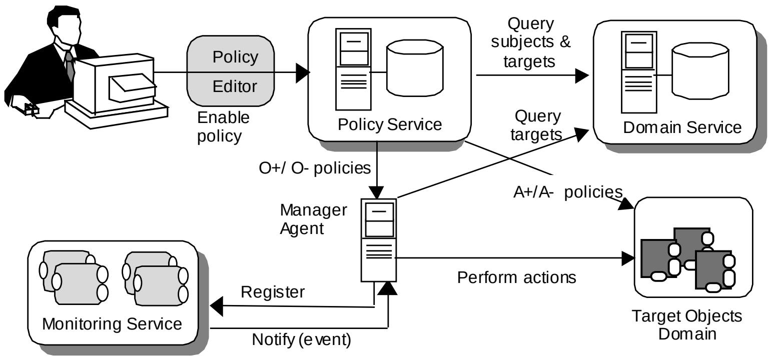

Fig. 2 Policy Enforcement

An overview of the approach to policy enforcement is given in Fig. 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 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 negative obligations restrain it.

Fig. 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 [12], [13], [14].

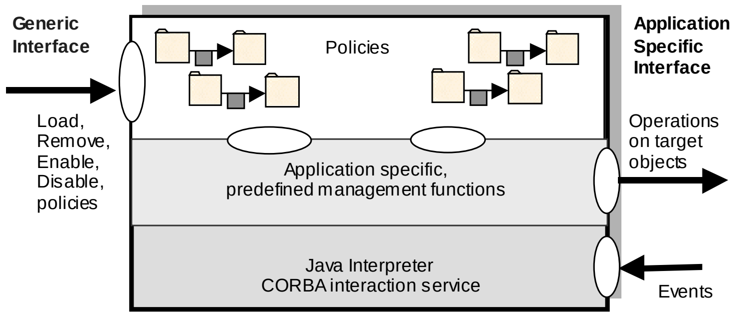
Generic Interface
Load, Remove, Enable, Disable, policies
Policies
Application specific, predefined management functions
Java Interpreter
CORBA interaction service
Application Specific Interface
Operations on target objects
Events

Fig. 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 [15].

Modality Conflicts – are inconsistencies which may arise when several policies with modalities of opposite sign refer to the *same* subjects, actions and targets. Therefore, these conflicts can be determined by syntactic analysis of polices. There are three types of modality conflicts:

- **O+/O-** subjects are both required and required not to perform the same actions on the target objects.
- **A+/A-** subjects are both authorised and forbidden to perform the actions on the target objects.
- **O+/A-** 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 X5 in Section 3.

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. Our tools analyse the policies within a domain to indicate conflicts for an administrator to resolve and allow precedence to be 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.

Semantic Conflicts and Metapolicies – while modality conflicts can be detected purely by syntactic analysis, application-specific conflicts arise from the semantics of the policies. For example, a conflict may arise if there are two policies which increase and decrease bandwidth allocation when the same event occurs. Similarly, 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. 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.

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. 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, and the set of authorisation and obligation policies defining the rights and duties for that position. Organisational positions can be represented as domains and we consider a *role* to be the set of policies (the arrows in Fig. 4) 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 [17].

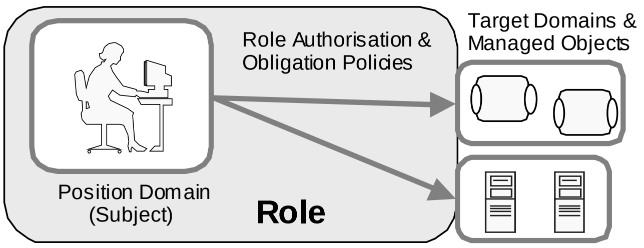

Fig. 4 Management Roles

Although the concept of role was originally defined to apply to people, it can also be used to group the authorisation and obligation policies that 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 providing 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 set of policies which are loaded onto that router. Another example is a mobile agent which is assigned to a visiting agent role 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 [18], [19]. These define inter-role relationships in terms of interaction protocols and concurrency constraints on the ordering of obligation actions. Furthermore, an object model for the specification of policy templates and role classes which uses inheritance to implement specialisation has also been defined. However, these issues will not be discussed further in this paper.

7 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 {/* provide adequate video conference set up */}
    @/users/groupA when 14:00 < time < 15:00

Network managers must provide 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 policy T1 into bandwidth management policies, authorisation policies and further administrative policies to enable or disable special policies which might apply during these hours. For example:

Administrative policies

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

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

Network managers must enable at 13:55 (T2) and disable at 15:00 (T3) special bandwidth control and QoS monitoring policies.

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/BandwidthControl + @/policies/QoSmonitoring

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

Bandwidth Control

T6 O+ on req(bw,chanId) edgeRouter {reduceReservation(bw)} channels/chanId
when bw < getReservation(chanId)

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

T7 O+ on req(bw, chanId) edgeRouter {increaseReservation(min(bw, x))}
channels/chanId when bw > getReservation(chanId)

Edge routers should increase bandwidth when the request is for more than the amount currently reserved. However, the amount reserved should not exceed x.

The refinement of abstract policies into implementable ones must be done by human managers. A positive obligation policy requires related authorisation policies giving subjects the necessary access rights to perform their tasks. 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. Thus the refinement of a policy does not preserve the policy modality or necessarily apply to the same 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 (T7, T8).

We currently maintain pointers from an abstract policy to the policies, derived from it, (omitted from the above examples for clarity) but we do not have tools to support the refinement process. We are investigating the use of requirements engineering tools and techniques for refinement and analysis of policies.

8 Related Work

There are a number of groups working on policies for network and distributed systems management [9],[20],[21]. Some of this has been based on our early proposals for policy notation. Another approach is to define policies using the full power of a general purpose scripting or interpreted language (e.g., TCL) and load this into network components. Bos [22] 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 [7]. A policy client (e.g. a router) makes policy requests on a server which retrieves the policy objects, interprets them and responds with policy decisions to the client. The client enforces the policy by, for example, permitting/forbidding requests 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 authorisations for network resources and services [23], [24], [25]. 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 *conditions* becomes true. These conditions correspond to a combination of our events and constraints for obligation policies. Their policy may be an aggregation of policy rules. They have realised policy conflicts can occur, but have not distinguished between modality and semantic conflicts nor do they say how conflicts will be detected. Directories are used for storing policies but not for grouping subjects and targets. They use dynamic groups 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 network device serves. Roles enable administrators to group the interfaces of multiple devices 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.

9 Conclusions

We have shown that our management policy and role approach, 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 network components. We have a prototype 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.

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.

References

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

2. Tennenhouse D, Smith J, Sincoskie D, Wetherall D, Minden G.: A survey of Active Network Research. IEEE Communications Magazine, 35(1):80-86, 1997.
3. Bieszczad A, Pagurek B, White T.: Mobile Agents for Network Management. *IEEE Communications Surveys*, 1(1), 1998. www.comsoc.org/pubs/surveys.
4. de Meer, et al.: Agents for Enhanced Internet QoS. *IEEE Concurrency* 6(2):30-39, 1998.
5. Lazar, A.: Programming Telecommunication Networks. *IEEE Network*, Sep/Oct 1997, 8-18
6. Goldszmidt, G., Yemini Y.: Evaluating Management Decisions via Delegation. In Hegering H, Yemini Y (eds.) Integrated Network Management III, Elsevier Science Publisher (1993), 247-257.
7. 3COM: Directory Enabled Networking and 3COM's Framework for Policy Powered Networking. from http://www.3com.com/,1998.
8. Sloman, M.: Policy Driven Management for Distributed Systems. *Journal of Network and Systems Management*, 2(4):333–360, Plenum Press, 1994.
9. Magee J., Moffett J. (eds.): Special Issue of *IEE/BCS/IOP Distributed Systems Engineering Journal* on Services for Managing Distributed Systems, **3**(2), 1996.
10. Sloman, M., Twidle, K.: Domains: A Framework for Structuring Management Policy. In Sloman M. (ed.): *Network & Distributed Systems Management*. Addison-Wesley (1994), 433–453.
11. Whal, M., Howes, T.,Kille S.: Lightweight Directory Access Protocol (v3), IETF RFC 2251, Dec. 1997. Available from http://www.ietf.org
12. Marriott, D., Sloman, M.: Management Policy Service for Distributed Systems. 3rd IEEE Int. Workshop on Services in Distributed and Networked Environments, Macau, 2–9, 1996.
13. Marriott, D., Sloman, M.: Implementation of a Management Agent for Interpreting Obligation Policy. IEEE/IFIP Distributed Systems Operations and Management Workshop (DSOM' 96), L'Aquila (Italy), Oct. 1996.
14. Marriott, D.: Management Policy for Distributed Systems. Ph.D. Dissertation, Imperial College, Department of Computing, London, UK, July 1997.
15. Lupu, E., Sloman, M.: Conflicts in Policy-Based Distributed Systems Management. To appear in IEEE Trans. on Soft. Eng., Special Issue on Inconsistency Management, 1999.
16. Mansouri-Samani M., Sloman, M.: GEM: A Generalised Event Monitoring Language for Distributed Systems. *IEE/BCS/IOP Distributed Systems Engineering*, **4**(2):96-108, 1997.
17. Lupu, E., Sloman, M.: Towards a Role-based Framework for Distributed Systems Management. *Journal of Network and Systems Management*, **5**(1):5-30,Plenum-Press, 1997
18. Lupu E., Sloman, M.: A Policy-based Role Object Model. 1st IEEE Enterprise Distributed Object Computing Workshop (EDOC'97*)*, Gold Coast, Australia, Oct.97, pp. 36-47.
19. Lupu, E.: A Role-Based Framework for Distributed Systems Management. Ph.D. Dissertation, Imperial College, Dept. of Computing, London, U.K, July 1998.
20. Koch, T. et al.: Policy Definition Language for Automated Management of Distributed System. 2nd IEEE Int. Workshop on Systems Management, Toronto, June 1996, 55-64.
21. Wies R.: 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.
22. Bos H.: Application Specific Policies: Beyond the Domain Boundaries. IFIP/IEEE Integrated Management Symposium (IM'99), Boston, May 1999.
23. Strassner J. Elleson, E.: Terminology for Describing Network Policy and Services, IETF draft work in progress, Feb. 1999. Available from http://www.ietf.org
24. Strassner J. Elleson, E., Moore, B.: Policy Framework Core Information Model, IETF draft work in progress, Feb. 1999, Available from http://www.ietf.org
25. Strassner J., Schleimer, S.: Policy Framework Definition Language, IETF draft work in progress, Nov. 1998, Available from http://www.ietf.org