Invited Paper to be published in IEEE Network Special issue on Policy Based Networking March 2002

Security and Management Policy Specification

Morris Sloman, Emil Lupu
Department of Computing, Imperial College, 180 Queen's Gate, London SW7 2AZ
m.sloman@doc.ic.ac.uk, e.c.lupu@doc.ic.ac.uk

Abstract

Policies are rules governing the choices in behaviour of a system. They are increasingly being used as a means of implementing flexible and adaptive systems for management of internet services, networks, and security systems. There is also a need for a common specification of security policy for large-scale, multi-organisational systems where access control is implemented in a variety of heterogeneous components. In this paper we survey both security and management policy specification approaches, concentrating on practical systems in which the policy specification can be directly translated into an implementation.

Keywords: policy-based management, security policy, security management, authorisation policy, obligation policy, roles, adaptive systems.

1 INTRODUCTION

Modern network components such as routers and switches need to be programmable to support the adaptive quality of service (QoS) required by multimedia applications and mobile computing users. Portable intelligent communicators will always have limited processing, storage, communication capability and battery life compared to desktop workstations and so will need to make use of local network services to provide a seamless ubiquitous computing environment. This environment will require fast service creation and resource management through a combination of network-aware applications and application-aware networks. Adaptive networks must support rapid deployment of customised services tailored for potentially mobile, corporate and individual users.

There are many mechanisms being promoted for programming network components, including code carrying IP packets which are executed by the routers traversed by the packets; scripts or interpreted code loaded via a management interface and mobile agents carrying both code and data which autonomously migrate around the network [1]. All of these are very powerful mechanisms which can potentially destroy the network if the code contains malicious or inadvertent bugs. Policies provide a more constrained means for adaptive behaviour of components [2].

Internet and e-commerce applications typically involve many different organisations – manufacturers, merchants, network service providers, banks and customers. Specifying and analysing the security for such environments to determine who can access what resources or information can be extremely difficult. The problem is that the access control is distributed across many heterogeneous components such as databases, operating systems, firewalls, and filtering routers controlled by the different interacting organisations. Even in the programmable network environment there is a need to define who is authorised and under what conditions they can inject code packets, scripts, mobile agents or policy rules into network components and what resources or functions these ‘programs’ can access. Security management needs similar adaptivity to that of network management to specify actions to take in response to simple security violations such as excessive login attempts but also for changing the behaviour and applicable policies in firewalls or web servers under denial of service or other network based attacks.

Policies are rules governing the choices in behaviour of a system [3]. *Obligation policies* are event triggered condition-action rules which can be used to define the conditions for reserving network resources, changing queuing strategy, loading code onto a router, or reconfiguring an FPGA for higher performance but with reduced error correction capability. Some policies may be user or application specific, such as what information to filter when bandwidth or device capabilities are limited. *Authorisation policies* are used to define what services or resources a subject (management agent, user or role) can access. Policies are persistent so that a one-off command to perform an action is not a policy. Scripts and mobile agents are often based on powerful interpreted languages such as Java, so can be used to introduce new functionality into network components. Policies define choices in behaviour in terms of the conditions under which predefined operations or actions can be invoked rather than changing the functionality of the actual operations themselves.

The main motivation for the recent interest in policy-based services, networks and security systems is to support dynamic adaptability of behaviour by changing policy without recoding or stopping the system [4]. This implies that it should be possible to dynamically update the policy rules interpreted by distributed entities to modify their behaviour.

Large-scale systems may contain millions of users and resources. It is not practical to specify policies relating to individual entities – instead, it must be possible to specify policies relating to groups of entities and also to nested groups such as sections within departments, within sites in different countries in an international organisation. It is also useful to group the policies pertaining to the rights and duties of a role or position within an organisation such as a network operator, nurse in a ward or mobile computing ‘visitor’ in a hotel.

Policies are derived from business goals, service level agreements or trust relationships within or between enterprises. The refinement of these abstract policies into policies relating to specific services and then into policies implementable by specific devices supporting the service is not easy, and not amenable to automation.

This paper provides a survey of some of the work on policy specification for both security management and policy-driven network management. Rather than covering many papers superficially we have concentrated on a few exemplary approaches in more detail, with emphasis on practical rather than theoretical approaches. In section 2 we describe examples of security policy specification – Role Based Access Control provides the grouping of policies (permissions) related to an organisational position and the Trust Policy Language indicates the use of credential-based policies. In section 3 we cover various approaches to management policy specification, namely the Policy Description Language for event-triggered policies from Lucent and the ongoing work in the IETF/DMTF on standardisation of policy information models. In section 4 we describe our Ponder policy specification language which combines many of the above concepts and can be used for both security and management policies. We do not discuss routing policies as these have been described in a recent survey [5]. WWW references to much of the work on policy and many of the papers described here can be found from [6].

2 SECURITY POLICY SPECIFICATION

2.1 Role-Based Access Control

Although Role-Based Access Control (RBAC) is not directly concerned with policy specification, it has been accepted as a security model which permits the specification and enforcement of organisational access control policies. The fundamental concept on which RBAC relies is that permissions are associated with roles rather than users, thus separating the assignment of users to roles from the assignment of permissions to roles. Users acquire access rights by virtue of their role memberships, and they can be dynamically assigned or removed from roles without changing the permissions associated with their role. Multiple users can be assigned to the same role and multiple roles can be assigned to the same users.

Although the concept of role has existed in a fairly similar form for a long time both in systems security and in Role Theory, the work presented by Sandhu in [7] and the first RBAC workshop [8] have prompted a renewed interest in this approach which is now adopted to different degrees in many commercial products.

The main goal of RBAC goes beyond the concept of role and aims to simplify permission management in large organisations. To achieve this, roles must be combined in a structured way and permissions must be "reused". The most popular approach relies on role inheritance where senior roles such as team leader, or project supervisor inherit the permissions of junior roles such as employee, team member, etc. However other approaches such as assigning roles to other roles can also be found in the literature.

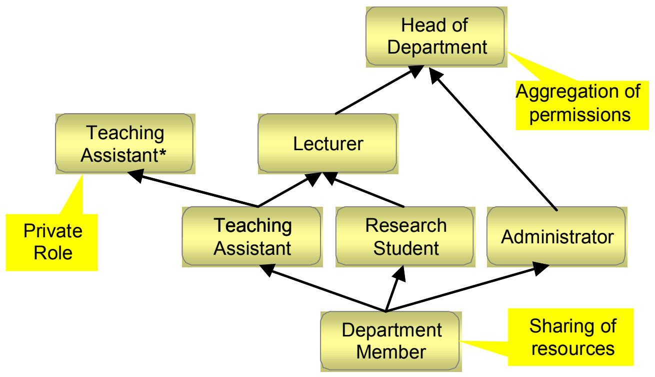

Figure 1 Role Hierarchies

A possible role hierarchy for an academic institution is described in Figure 1 which illustrates how role inheritance provides the sharing of resources through common lower level roles and the aggregation of permissions through inheritance of permissions to the higher level. However, such role inheritance hierarchies are not without shortcomings as there are numerous exceptions to the rule that senior roles inherit all the permissions of junior roles. Most notably, access to private files and permissions granted by virtue of a competency are not inherited. For example, the head of department does not usually inherit the access right of a system administrator. To accommodate such situations it is necessary to create private roles as shown in Figure 1, which group the permissions that are not inherited upwards in the hierarchy. Implementing an RBAC system with inheritance of permissions between roles considerably reduces the number of permissions in the system. However, in a distributed system it may also render access control checks, performed on each invocation, more complex since the inherited roles may be stored remotely and checking the inherited permissions may require several remote invocations. To avoid this increased complexity, a capability-based system may be more appropriate for RBAC since it shifts the responsibility for collecting the inherited permissions to the user (subject) system and this is done prior to the access control check.

Several constraints may apply to an RBAC model across its associations, between users and roles, roles and permissions or between roles themselves (inheritance) [9]. Amongst these, separation of duty constraints, which identify mutually exclusive sets of permissions that a user is not allowed to hold, have been the subject of the most intensive work.

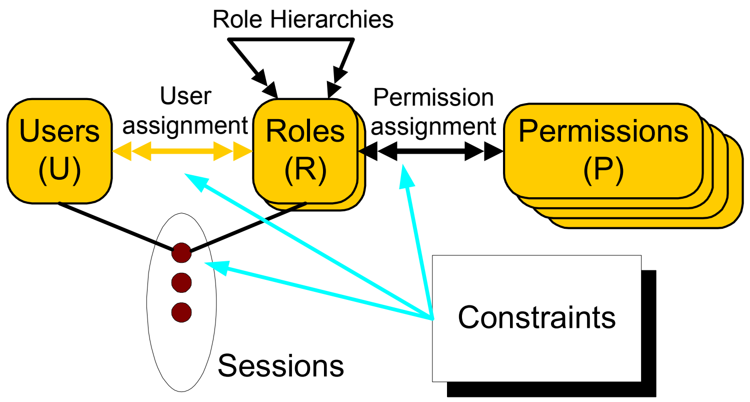

Figure 2 RBAC Model (From [7])

Figure 2 shows an overview of the RBAC model with the various relationships between users roles and permissions, and the constraints. The model presented in [7] also introduces the concept of sessions. A session groups the permissions from a selected number of roles which the user may want active at a given moment in time i.e., in a work session. The user can then use all access rights from the active roles in order to carry out his tasks.

The RBAC model presented above is the foundation from which many variations have developed. In particular, the team-based access control has developed as a means of simultaneously activating a set of related roles e.g., the surgeon, nurses and other personnel in an operating theatre.

Although, the concept of role is not new, the introduction of RBAC models has fostered a change from the traditional mandatory and discretionary access control models to a new framework where the access control policy is neither rigidly embedded in the implementation nor left to the owner of each resource but can be implemented on the basis of clearly specified organisational policies.

2.2 IBM's Trust Policy Language

There are a number of groups which use the term 'trust management' for frameworks that support sophisticated authorisation policy specification and implementation using public key certificates as credentials to authenticate identities or membership of groups [10], [11]. This can be particularly useful for e-commerce and internet applications. We will describe the IBM Trust Policy Language (TPL) and support system as an exemplar of this approach [12].

TPL uses XML to define policy rules which specify the criteria for a client to be assigned to a group which is similar to a role. Standard database or operating system authorisation rules are then used to define what resources a group can access as shown in Figure 3. Thus, the authority that issues certificates for a role can be completely different from the one that defines access control policy for a resource.

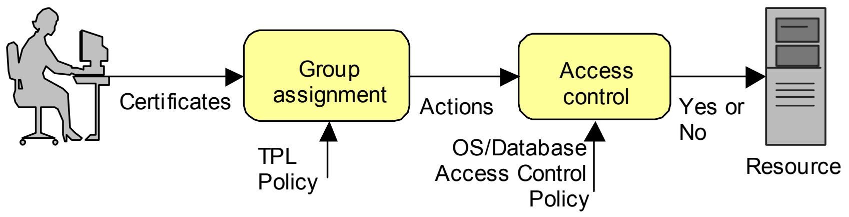

Figure 3 Separation of Group Assignment from Access Control Decision

The TPL policy consists of a list of required X509 certificates which permit a client to become a member of a group and a Boolean function relating the values of the fields in one or more certificates to define group membership criteria. Example groups include employees of an organisation, registered customer for a merchant or a recognised cardiologist at a hospital. An example medical group membership criteria could be possession of 3 recommendation certificates from hospitals with recommendation level >3.

A certificate identifies the issuer, the subject to whom they are issued (which could be a name or an anonymous public key) and a list of name value pairs which may contain information such as age, sex and rank of the holder. The certificate may also contain links to repositories for additional certificates for the issuer or subject as multiple certificates may be needed for a client to be assigned to a group.

```
<POLICY>

    <GROUP NAME=”self”>
    </GROUP>

    <GROUP NAME=”partners”>
        <RULE>
            <INCLUSION ID=”partner” TYPE=”partner” FROM “self”>
            </INCLUSION>
        </RULE>
    </GROUP>

    <GROUP NAME=”departments”>
        <RULE>
            <INCLUSION ID=”partner” TYPE=”partner” FROM=”partners”>
            </INCLUSION>
        </RULE>
    </GROUP>

    <GROUP NAME=”customers”>
        <RULE>
            <INCLUSION ID=”customer” TYPE=”employee” FROM=”departments”>
            </INCLUSION>
            <FUNCTION>
                <GT>
                    <FIELD ID=”customer” NAME=”rank”></FIELD>
                    <CONST>3</CONST>
                </GT>
            </FUNCTION>
        </RULE>
    </GROUP>

</POLICY>
```

Figure 4 TPL Example

The TPL policies shown in Figure 4 are for a *retailer* (the *self* group in the policy) to give discounts to preferred customers who are employees of a department of a partner company. Entities that have *partner* certificates, signed by the retailer, are placed in the group partners. The group *department* is defined as any user having a *partner* certificate signed by the partners group. Finally, the *customer* group consists of anyone that has an *employee* certificate signed by a member of the departments group who has a rank > 3.

TPL is available for download from IBM Alphaworks and is very flexible as a means for defining authorisation policy for an internet service but the XML syntax is rather verbose and unreadable. Although a policy can have multiple rules, there is no inheritance or reuse between different policy specifications and it is not suitable for specifying security management policy.

2.3 Other Security Policy Specification Approaches

There has been considerable interest in logic-based approaches to specifying authorisation policy as exemplified by [13], [14]. They assume a strong mathematical background, which can make them difficult to use and understand, and they do not easily map onto implementation mechanisms. The ASL language [13] includes a form of meta-policies called integrity rules to specify application-dependent rules that limit the range of acceptable access control policies. Although it provides support for role-based access control, the language does not scale well to large systems because there is no way of grouping rules into structures for reusability. A separate rule must be specified for each action. There is no explicit specification of delegation and no means of specifying authorisation rules for groups of objects that are not related by type.

Ortalo [14] describes a language to express security policies in information systems based on the logic of permissions and obligations, a type of modal logic called *deontic logic*. Standard deontic logic centres on impersonal statements of the form "it is obliged that p" which do not necessarily identify to whom the obligation applies. Ortalo accepts the axiom $Pp = \neg O \neg p$ ("permitted p is equivalent to not p being not obliged") as a suitable definition of permission. In our view, an obligation policy requires a relevant authorisation policy to permit the actions defined in the obligation but an obligation policy does not imply an authorisation policy.

Others, focus on the specification and implementation of access control policies for mobile agent systems. In most cases the studies focus on reconfigurable access control policies in the Java environment. They have included both the translation of higher level policies into Java security policies as well as different access control mechanisms within Java [15], [16].

3 MANAGEMENT POLICY SPECIFICATION

3.1 Lucent's Policy Definition Language

The Policy Definition Language (PDL) was developed at Lucent Bell Labs for network management and is based on active database declarative event-condition-action rules i.e., obligation policies [17]. It is a very simple language with 2 main constructs – a policy rule corresponding to an obligation policy and a rule for triggering other events:

Policy Rule: *event* ***causes*** *action* $(t_1 = v_1, \ldots. t_k = v_k)$ ***if*** *condition*
Policy Defined Event (pde): *event* ***triggers*** *pde* $(t_1 = v_1, \ldots. t_k = v_k)$ ***if*** *condition*

where t_i = parameter (attribute) type and v_i = value. Every event has a timestamp and the URL of the source which generated it. Primitive events can be generated by the managed environment or by other PDEs. An Epoch is an application specific time window in which a set of events are considered to occur simultaneously e.g., day, hour, second. Complex events include:

- e1 & e2 &… & en = conjunction of events in an epoch
- e1 | e2 |… | en = disjunction of events in an epoch
- !e = no occurrence of event in an epoch
- ^e = a sequence of zero or more occurrences of an event
- group (e1 & e2 &… & en) or group (e1 | e2 |… | en) =
  a single event grouping all instances of the complex event. If there are n instance of e1 and m instances of event e2 in an epoch then there is only one occurrence of group (e1 & e2) instead of m x n occurrences.
- count (e, e.x>4) counts the number of occurrences of event e for which parameter x > 4 is true
- Other aggregations – max, min, ave etc. relating to event parameters.

An example is given in Figure 5 of the use of PDL to define a policy for a network component which rejects call requests when in overload mode.

```
Overload threshold: (time outs/calls made) > t
Normal threshold: (time outs/calls made) < n

Events:     normal, restricted : pde
            callMade, timeOut, powerOn: system events
            /* the above events have no attributes */

Actions:    restrictCalls, acceptAll

Policies:   powerOn triggers normal  // a normal event is triggered when the component is switched

            normal, ^(callMade | timeOut) triggers restricted if count (timeOut) > t*count(callMade)
            /* after a normal event, a sequence of callMade or timout events will trigger restrictCalls if the
            overload threshold is exceeded */

            restricted causes restrictCalls   // sets overload mode to reject call requests

            restricted, ^(callMade | timeout) triggers normal if count (timeOut) < n*count(callMade)
            /* when in overload mode, a sequence of callMade or timout events will trigger normal if the
            normal threshold is exceeded */

            normal causes acceptAll

            /* Assumes only one callMade or timeOut event per epoch */
```

Figure 5 PDL Example

PDL is implemented and used in Lucent switching products [18]. It shows that event-condition-action rules are a very flexible approach to specifying management policy. It has also been extended to specify complex workflow tasks. However, it does not support any form of policy composition or reuse of specifications.

3.2 CIM Policy model

The Distributed Management Task Force (DMTF) [19], in collaboration with the Internet Engineering Task Force (IETF) Policy work group [20], are defining a policy information model as an extension to the Core Information Model (CIM), known as PCIM [21]. An information model is "an abstraction and representation of the entities in a managed environment – their properties, operation, and relationships". This is independent of any specific repository, application, protocol or platform. The IETF are defining a mapping of the PCIM model to a directory schema so that an LDAP (Lightweight Directory Access Protocol) directory can be used as a repository. The CIM defines generic objects such a managed system elements, logical and physical elements, systems, service and service access point etc. The Policy Model defines a policy rule and its component policy conditions and policy actions as shown in Figure 6 and 7. The assumption is that a policy rule is of the form if <condition set> then do <action list>. The condition set can be expressed in either disjunctive or conjunctive normal form. Note that there is no explicit triggering event specified as part of the policy, although it is similar to an obligation policy. It is assumed that the agent interpreting the event will evaluate the policy when an implicit event occurs such as a new session is set up or possibly on every message.

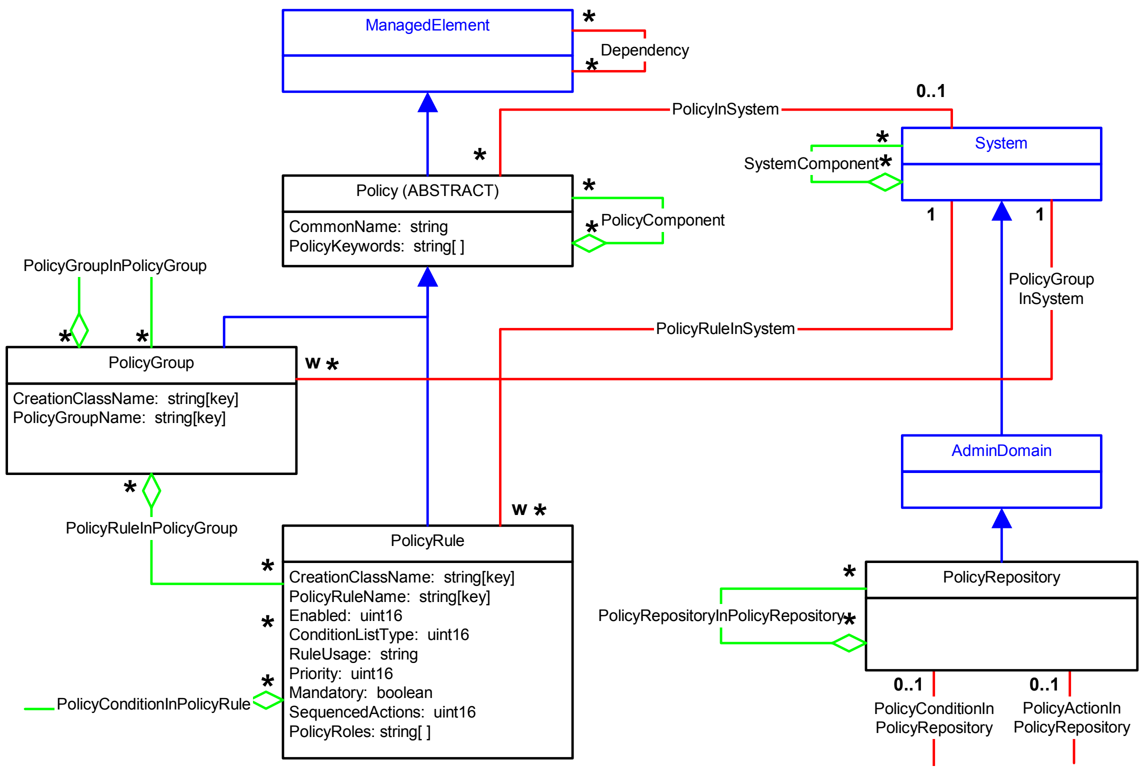

Figure 6 (PCIM) Policy Core Information Model (from [23])

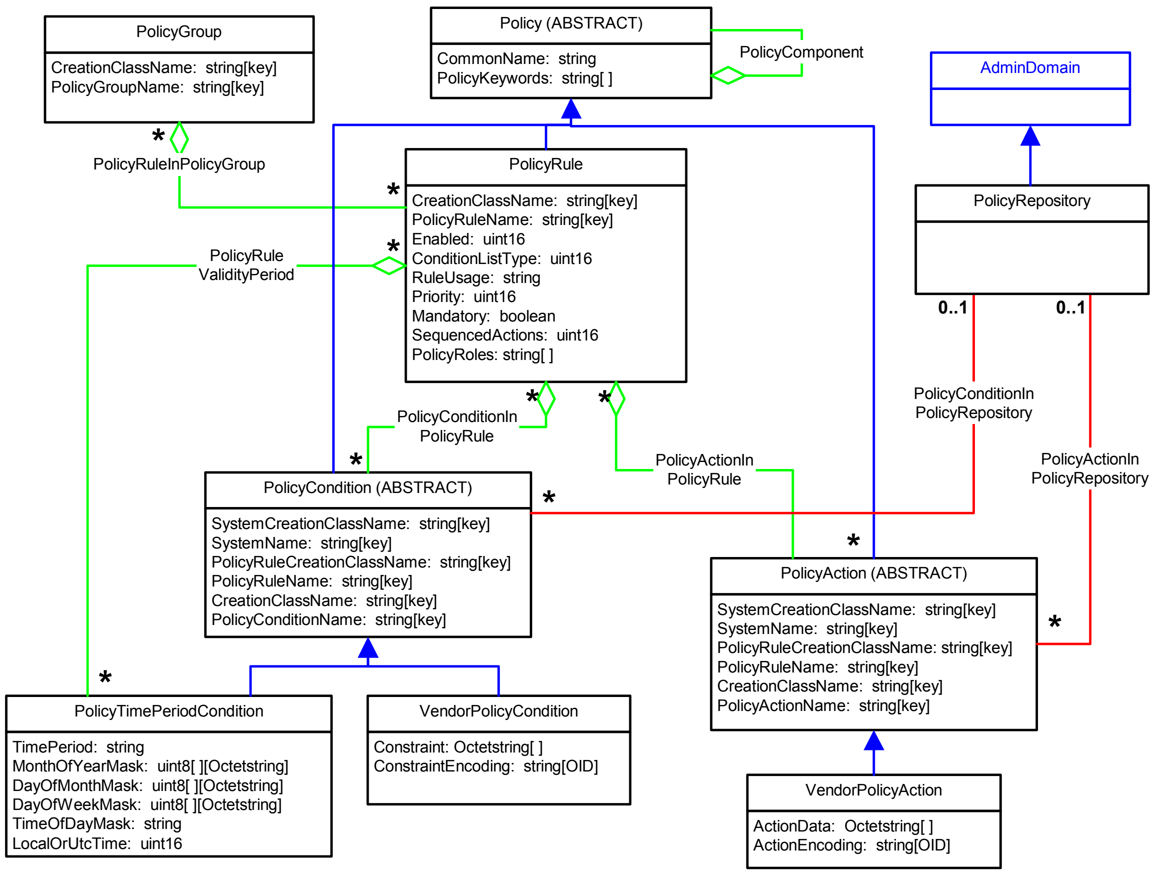
PolicyGroup
CreationClassName: string[key]
PolicyGroupName: string[key]
Policy (ABSTRACT)
CommonName: string
PolicyKeywords: string[ ]
PolicyComponent
AdminDomain
PolicyRuleInPolicyGroup
PolicyRule
CreationClassName: string[key]
PolicyRuleName: string[key]
Enabled: uint16
ConditionListType: uint16
RuleUsage: string
Priority: uint16
Mandatory: boolean
SequencedActions: uint16
PolicyRoles: string[ ]
PolicyRule ValidityPeriod
PolicyRepository
0..1
0..1
PolicyConditionIn PolicyRepository
PolicyConditionIn PolicyRule
PolicyActionIn PolicyRule
PolicyActionIn PolicyRepository
PolicyCondition (ABSTRACT)
SystemCreationClassName: string[key]
SystemName: string[key]
PolicyRuleCreationClassName: string[key]
PolicyRuleName: string[key]
CreationClassName: string[key]
PolicyConditionName: string[key]
PolicyAction (ABSTRACT)
SystemCreationClassName: string[key]
SystemName: string[key]
PolicyRuleCreationClassName: string[key]
PolicyRuleName: string[key]
CreationClassName: string[key]
PolicyActionName: string[key]
PolicyTimePeriodCondition
TimePeriod: string
MonthOfYearMask: uint8[ ][Octetstring]
DayOfMonthMask: uint8[ ][Octetstring]
DayOfWeekMask: uint8[ ][Octetstring]
TimeOfDayMask: string
LocalOrUtcTime: uint16
VendorPolicyCondition
Constraint: Octetstring[ ]
ConstraintEncoding: string[OID]
VendorPolicyAction
ActionData: Octetstring[ ]
ActionEncoding: string[OID]
*

Figure 7 PCIM Conditions and Actions (from [23])

Policy rules may be aggregated into nested policy groups to define the policies pertaining to a department, user etc. Conditions and actions may be specific to a rule or can optionally be stored separately in a policy repository and reused by multiple rules. Sophisticated time period conditions can be defined in terms of times, masks for days in a week, days at beginning or end of month, months in year etc. The actions can be defined as being sequential or any order.

The IETF has also defined extensions to PCIM called the QoS Policy Information Model (QPIM) which contains a set of abstractions specific to IntServ and DiffServ management [22]. For example they have abstract objects for classifiers, meters, shapers, droppers and queues, which can be mapped onto the control elements provided by a specific DiffServ router. Specific priority values are assigned to policy rules to resolve conflicts.

A simple DiffServ QPIM Policy is shown in Figure 8.

```
If (SourcePort == MyWebServerPort) then Color DSCP=5

This is represented by the following LDAP Objects

Objectclass:qosPolicyRule
        Type: 1
        Direction: out
        Priority: 1

        Objectclass: qosColorPolicyAction
                DSCPValue: 5

        Objectclass: qosColorPolicyCondition
                Type: Integer OID
                Operator: "=="

                Objectclass: qosPolicyVariable
                        Name: SourcePort
                        Type: IntegerOID

                Objectclass: qosPolicyConstant
                        Name: MyWebServerPort
                        Type: IntegerOID

                        Objectclass: qosPolicyNumberValue
                                Type: IntegerOID
                                PortValue: 80
```

Figure 8 QPIM Example Policy (from [23])

As can be seen from Figure 8, the LDAP schema representation of a very simple policy is extremely verbose and not really aimed at human interpretation. It is assumed this will be generated from a graphical tool of the form provided by a number of vendors or from a high level language such as Ponder, described in section 4. PCIM does not distinguish between authorisation and obligation policies. The emphasis in the working groups has been on QoS policies which can be considered obligation policies without event triggers. However they can specify simple admission control policies by means of an action to accept or deny a message to give the effect of a positive or negative authorisation policy. The term 'role' is defined as a characteristic of a managed element and is used as a means of identifying elements to which a policy applies e.g., a *gigabitEthernet* role policy may apply to all elements which have a gigabit Ethernet interface.

It is assumed that policies are objects stored in an LDAP directory service. A policy consumer (policy decision point – PDP) retrieves policies from the policy repository (e.g., LDAP server). A policy execution point (PEP), such as a router, requests policy decisions using the Common Open Policy Service Protocol (COPS). The PEP enforces the policy, for example, by permitting/forbidding requests or allocating packets from a connection to a particular queue. A PEP and PDP could be combined into a single component. Note that there is no explicit event to trigger an IETF policy. An implicit event such a packet arrival at a router will trigger the search for applicable policies. The policy condition will typically define a specific address or range of source or destination IP addresses or ports to which a policy applies so this information taken from the packet can be used to find which policies apply. It is only practical for the PEP to query the PDP for a decision on comparatively infrequent packets such as an IntServ request related to a connection set up. If policy decisions are needed on every packet, such as for DiffServ, then these have to be preloaded into the PEP by the PDP using the COPS policy provisioning mode. Note that the use of

COPS is not mandated by the IETF and other protocols such as SNMP or HTTP could be used for transferring policy information. There have been suggestions for extensions to the IETF approach to include explicit events for policies that deal with failures, overload situation etc., but so far nothing concrete has emerged.

3.3 Other Approaches to Policy Specification

There are a number of vendors producing policy-based management tools, usually with a graphical user interface for defining policies and allocating them to devices to which they apply. These generally follow the approach of the IETF policy work, which is still in progress, so there are not yet any standards to be compliant with. Initially the main focus was on QoS management but some of the vendors are extending this to include support for provisioning of virtual private networks (see [6] for links to vendor web sites). [24] gives a detailed description of the architectures and algorithms needed to support an IETF approach to policy-based networking. It covers some of the issues of translating from higher level service level goals into policies for specific components in the network and how to distribute the policies to the relevant components.

The ODP Reference Model Enterprise Viewpoint [25] defines concepts for specifying an abstraction of a system within a defined environment in terms of the system's purpose, its scope and the policies that apply to the system from its environment as well as those defined within the system. The current specification defines terms such as policy, authorisation, prohibition, obligation, role, domain, and community but there is no associated language by which a system can be clearly specified, analysed or implemented. There have been several attempts at defining Enterprise Viewpoint specifications by using the Unified Modelling Language UML possibly with added extensions [26],[27],[28]. Proposals have alternated between modelling policies as notes attached to the design elements, OCL constraints or additional extensions to UML. [27] indicates that OCL is not directly suitable for expressing ODP prohibitions or obligations. These approaches encounter difficulties related to the level of abstraction of the specification. Enterprise Viewpoint specifications are intended to be more general and more abstract than the system design elements of UML. Hence, Enterprise Viewpoint concepts are not defined in UML but can be realised (implemented) by a design that could be specified in UML. For example, an authorisation policy is implemented by a set of objects such as capabilities, certificates or access control list entries, and an obligation can be realised by the implementation of a particular activity diagram or collaboration. However, the enterprise level description requires a more abstract representation of the policy than the diagram of invocations, which implement it. It is therefore important to be able to identify a minimal and concrete representation for the Enterprise Viewpoint concepts, which represent functional aspects of the system rather than encumber a design notation with abstract concepts that need to be implemented by other representations in the same notation. The various models proposed for defining Enterprise Concepts in UML are complex and differ substantially between them with little prospect of convergence or consensus.

Minsky's "law governed systems" specify permissions and prohibition as a set of rules which are similar to positive and negative authorisations. Their approach supports a common global set of constraints, similar to obligation policies, which are implemented by means of filters in every node which check that all interactions are consistent with a global law [29].

4 PONDER

The Ponder language for specifying Management and Security policies [30] evolved out of work on policy management at Imperial College over a period of about 10 years. Ponder is a declarative, object-oriented language that can be used to specify security policies that map onto various access control mechanisms for firewalls, operating systems, databases and Java [15]. It supports obligation policies that are event triggered condition-action rules for policy based management of networks and distributed systems. Ponder can also be used for security management activities such as registration of users or logging and auditing events for dealing with access to critical resources or security violations. Key concepts of the language include domains to group the objects to which policies apply, roles to group policies relating to a position in an organisation [31], relationships to define interactions between roles and management structures to define a configuration of roles and relationships pertaining to an organisational unit such as a department or a section.

4.1 Domains

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. 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 objects. A domain is thus very similar in concept to a file system directory but may hold references to any type of objects, 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 i.e., domains can overlap. 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. Domains have been implemented as directories in an extended LDAP Service.

4.2 Ponder primitive policies

Authorisation policies define what activities a member of the subject domain can perform on the set of objects in the target domain. These are essentially access control policies, to protect resources and services from unauthorized access. A positive authorisation policy defines the actions that subjects are permitted to perform on target objects. A negative authorisation policy specifies the actions that subjects are forbidden to perform on target objects.

The language provides reuse by supporting the definition of policy types to which any policy element can be passed as formal parameter. Multiple instances can then be created and tailored for the specific environment by passing actual parameters as shown in Figure 9.

```
type auth+ PolicyOpsT (subject s, target <PolicyT> t) {
            action load(), remove(), enable(), disable() ; }

inst auth+ switchPolicyOps=PolicyOpsT(/NetworkAdmins, Nregion/switches);
inst auth+ routersPolicyOps=PolicyOpsT(/QoSAdmins, /Nregion/routers);
```

The two policy instances created from a *PolicyOpsT* type allow members of /NetworkAdmins and /QoSAdmins (subjects) to load, remove, enable or disable objects of type PolicyT within the /Nregion/switches and /Nregion/routers domains (targets) respectively.

Figure 9 Example Ponder Authorisation Policies

Policies can also be declared directly without using a type as shown in the negative authorisation policy in Figure 10, which also shows the use of a time based constraint to limit the applicability of the policy

```
inst auth– /negativeAuth/testRouters {
      subject     /testEngineers/trainee ;
      action      performance_test() ;
      target      <routerT> /routers ;
      when        time.between (“0900”, “1700”)
}
```

Trainee test engineers are forbidden to perform performance tests on routers between the hours of 0900 and 1700. The policy is stored within the /negativeAuth domain.

Figure 10 Direct Policy Declaration

Ponder also supports a number of other basic policies for specifying security policy: *Information filtering* policy can be used to transform input or output parameters in an interaction. For example, a location service might permit access to detailed location information, such as the specific room in which a person is, only to users within the department. External users can only determine whether a person is at work or not. *Delegation* policy permits subjects to grant privileges, which they possess (due to an existing authorisation policy), to grantees to perform an action on their behalf e.g., passing read rights to a printer spooler in order to print a file. *Refrain* policies define the actions that subjects must refrain from performing (must not perform) on target objects even though they may actually be permitted to perform the action. Refrain policies act as restraints on the actions that subjects perform and are implemented by subjects. See [30] for more details and examples of these policies.

Obligation policies are event-triggered, condition-action rules similar to Lucent’s PDL, and define the activities subjects (human or automated manager components) must perform on objects in the target domain. Events can be simple, i.e. an internal timer event, or an external event notified by monitoring service components e.g. a temperature exceeding a threshold or a component failing. Composite events can be specified using event composition operators.

```
inst oblig loginFailure {
      on                3*loginfail(userid) ;
      subject           s = /NRegion/SecAdmin ;
      target <userT>    t = /NRegion/users ^ {userid} ;
      do                t.disable() -> s.log(userid) ;
}
```

This policy is triggered by 3 consecutive loginfail events with the same userid. The NRegion security administrator (SecAdmin) disables the user with userid in the /NRegion/users domain and then logs the failed userid by means of a local operation performed in the SecAdmin object. The ‘->’ operator is used to separate a sequence of actions in an obligation policy. Names are assigned to both the subject and the target. They can then be reused within the policy. In this example we use them to prefix the actions in order to indicate whether the action is on the interface of the target or local to the subject.

Figure 11 Example Ponder Obligation Policy

4.3 Ponder Composite Policies

Ponder composite policies facilitate policy management in large, complex enterprises. They provide the ability to group policies and structure them to reflect organisational structure, preserve the natural way system administrators operate or simply provide reusability of common definitions. This simplifies the task of policy administrators.

Roles provide a semantic grouping of policies with a common subject, generally pertaining to a position within an organisation such as department manager, project manager, analyst or ward-nurse. Specifying organizational policies for human managers in terms of manager positions rather than persons permits the assignment of a new person to the manager position without re-specifying the policies referring to the duties and authorizations of that position. A role can also specify the policies that apply to an automated component acting as a subject in the system. Organisational positions can be represented as domains and we consider a role to be the set of authorisation, obligation, refrain and delegation policies with the *subject domain* of the role as their subject. A role is just a group of policies in which all the policies have the same subject, which is defined implicitly, as shown in Figure 12.

```
type role ServiceEngineer (CallsDB callsDb) {
      inst oblig serviceComplaint {
            on          customerComplaint(mobileNo) ;
            do          t.checkSubscriberInfo(mobileNo) ->
                        t.checkPhoneCallList(mobileNo) ->
                        traceSignalReception(mobileNo);
            target      t = callsDb ; // calls register }

      inst oblig deactivateAccount { . . . }
      inst auth+ serviceActionsAuth { . . . }
      // other policies
}
```

The role type ServiceEngineer models a service engineer role in a mobile telecommunications service. A service engineer is responsible for responding to customer complaints and service requests. The role type is parameterised with the calls database, a database of subscribers in the system and their calls. The obligation policy serviceComplaint is triggered by a customerComplaint event with the mobile number of the customer given as an event attribute. On this event, the subject of the role must execute a sequence of actions on the calls-database in order check the information of the subscriber whose mobile-number was passed in through the complaint event, check the phone list and then trace the signal reception. Note that the obligation policy does not specify a subject as all policies within the role have the same implicit subject.

Figure 12 Example Role policy

Managers acting in organisational positions (roles) interact with each other. A *relationship* groups the policies defining the rights and duties of roles towards each other. It can also include policies related to resources that are shared by the roles within the relationship. It thus provides an abstraction for defining policies that are not the roles themselves but are part of the interaction between the roles. The syntax of a relationship is very similar to that of a role but a relationship can include definitions of the roles participating in the relationship. However roles cannot have nested role definitions. Participating roles can also be defined as parameters within a relationship type definition as shown below.

```
type rel ReportingT (ProjectManagerT pm, SecretaryT secr) {
        inst oblig reportWeekly {
                on              timer.day ("monday") ;
                subject         secr ;
                target          pm ;
                do              mailReport() ;
  }
        // . . . other policies
}
```

The ReportingT relationship type is specified between a ProjectManager role type and a Secretary role type. The obligation policy reportWeekly specifies that the subject of the SecretaryT role must mail a report to the subject of the ProjectManagerT role every Monday. The use of roles in place of subjects and targets implicitly refers to the subject of the corresponding role.

Figure 13 Example Relationship type

Many large organisations are structured into units such as branch offices, departments, and hospital wards, which have a similar configuration of roles and policies. Ponder supports the notion of *management structures* to define a configuration in terms of instances of roles, relationships and nested management structures relating to organisational units. For example a management structure *type* would be used to define a branch in a bank or a department in a university and then *instantiated* for particular branches or departments. A management structure is thus a composite policy containing the definition of roles, relationships and other nested management structures as well as instances of these composite policies.

Figure 14 shows a simple management structure for a software development company consisting of a project manager, software developers and a project contact secretary. Figure 15 gives the definition of the structure.

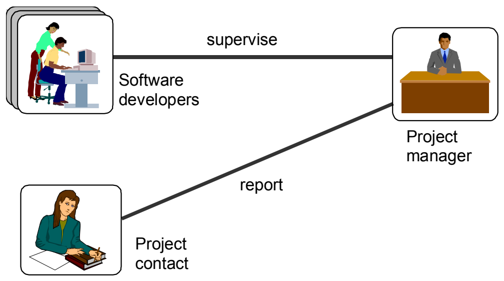

Figure 14 Simple Management Structure

```
type mstruct BranchT (...) {
        inst    role    projectManager      = ProjectManagerT(…);
                role    projectContact      = SecretaryT(...);
                role    softDeveloper       = SoftDeveloperT(...);

        inst    rel     supervise = SupervisionT (projectManager, softDeveloper);
                rel     report = ReportingT (projectContact, projectManager);
}
inst    mstruct branchA = BranchT(…);
        mstruct branchB = BranchT(…);
```

This declares instances of the 3 roles shown in Figure 12. Two relationships govern the interactions between these roles. A supervise relationship between the softDeveloper and the projectManager, and a reporting relationship between the ProjectContact and the projectManager. Two instances of the BranchT type are created for branches within the organisation that exhibit the same role-relationship requirements.

Figure 15 Software company management structure

Ponder allows specialisation of policy types, through inheritance. When a type extends another, it inherits all of its elements, adds new elements and overrides elements with the same name. This is particularly useful for specialisation of composite policies. For example it would be possible to define a new type of mobile systems project manager, from a project manager role (c.f. Figure 12) with additional policies.

In Ponder a person can be assigned to multiple roles but rights from one role cannot be used to perform actions relating to another role. A person can also have policies that pertain to him/her as an individual and have nothing to do with any roles. In RBAC inheritance is based on policy instances and all policies are defined in terms of roles. This means RBAC requires a much more complicated role structure to separate the policies that are inherited from those that are private.

A compiler has been implemented for the Ponder language. Various backends have also been implemented to generate firewall rules, Windows access control templates, Java security policies [15] and Java obligation policy rules for interpretation by a policy agent. We also have a system to automatically disseminate policies to the relevant agents which will interpret them i.e., to subjects for obligation and refrain policies and access control agents for authorisation and filter policies [33].

5 RESEARCH ISSUES

5.1 Conflict Analysis

There are a number of different conflicts that can arise from policies. For example, it might be possible that different QPIM policies apply for marking packets relating to a specific flow as one selects packets based on a range of IP addresses and another selects packets based on port numbers. The solution proposed in the IETF framework advocates assigning explicit priorities to policies but this is notoriously difficult to do in large systems where many different people are responsible for specifying policies. In Ponder modality conflicts arise between positive and negative policies which apply to the same subjects, targets and actions [32]. These can be detected by syntactic analysis of the policies as the conflict can be determined by detecting overlap of subjects, targets and actions. However, the analysis detects only potential conflicts rather than actual conflicts since constraints may limit the applicability of the policy to disjoint sets of circumstances e.g., different times of day. While modality conflicts can be detected syntactically, other conflicts can only be determined by understanding the actions being performed by the policies. For example, there will be a conflict between two policies which result in the same packet being placed on 2 different queues. Similarly, separation of duty conflicts arise from authorisation policies which permit the same person to approve payments and sign cheques. Generally, these conflicts are application specific and to detect them it is necessary to specify the conditions which result in conflict. The approach is therefore to specify constraints on the set of policies (i.e. meta-policies) using a suitable notation and then analyse the policy set against these constraints to determine if there are any conflicts [32]. Whether conflicts occur or not may depend on run-time parameters specified in constraints such as time or the current state of the components to which the policy apply. It is thus rather difficult to determine all possible conflicting conditions in advance and so it is still necessary to detect conflicts at run-time. Furthermore, when conflicting policies are detected it is not obvious how to resolve the conflicts automatically. Explicit priority may work in some cases. In some situations, negative authorisation policies should override positive ones, but in other situations the positive authorisation is an exception to a more general negative authorisation. In some situations more specific policies that apply to a department may override general policies applying to the whole organisation. We have been experimenting with meta-policies which define an application specific precedence relationships between conflicting policies.

Although some progress has been made in dealing with policy conflicts [24], [32], significant challenges remain to be addressed. In particular, how can one detect conflicts when arbitrary conditions restrict the applicability of the policies? Sometimes, it is possible to compare restrictions placed by the constraints. For example, it is possible to detect if two time intervals overlap or if the policies apply when subjects are in different states e.g., active or standby. However, the problem remains unsolved in the general case. Other challenges concern the different levels of abstraction at which policy is specified. Conflicts between organisational goals will inevitably lead to conflicts between the policies derived from these goals. Some policies will trigger complex management procedures which require the execution of actions that may be specified as part of different policies. This, renders the task of ensuring the consistency of a policy specification much more complex.

5.2 Refinement

Both network and security policies are specified with a view to fulfil organisational goals and service level objectives. The process of deriving a more concrete specification from a higher-level objective is termed refinement. Although, the goal of automating refinement of management and security policies from higher-level objectives remains a worthy one, it is not practical for all but the most trivial scenarios. However, this does not preclude the partial automation of this process or providing tools which can assist human managers to refine high-level abstract policies into more concrete ones. This will require representing and taking into account domain specific knowledge as well as more general techniques and methodologies. Currently the most promising approach seems to be investigated in requirements engineering and relies on identifying, recognising and instantiating *refinement patterns* [34]. A much simpler approach of integrating Service Level goals with policies is described in [35].

It is desirable to maintain the properties of *consistency* and *completeness* when refining an abstract goal into a set of more concrete policies. By consistency we mean that there is no conflict between those policies derived by refinement from a high-level goal nor are they in conflict with the other policies within the system. By completeness we mean that the set of policies derived by refinement fully implement the intended specification of the abstract goal. Using established patterns would help towards maintaining both these properties as patterns can be previously checked for completeness and for conflicts. Note, that this does not ensure that conflicts will not occur with other policies in the system thus even when instantiating established patterns it is necessary to perform conflict analysis.

5.3 Multiple levels of policy

Policies can be used to support adaptability at multiple levels in a network (i) within network-aware applications, (ii) within application-aware networks, and (iii) at the hardware level to support adaptability in the packet forwarding "fastpath" of network elements. Research is needed on defining interfaces for the exchange of policies between these levels. For example an application specific policy may be more efficiently interpreted within a network component or an application may need to adapt its behaviour as a result of adaptation within the network. However it is not easy to map the semantics of the policies between the different levels. The application may not be aware of what components exist within the network and so how can it specify policies to be interpreted by them?

An interesting variation of the above is to consider a policy feedback loop where the system is monitored to see whether it is performing according to high-level policies or to determine changes in the systems due to faults, new applications or users appearing and hence to dynamically modify the lower level policies in order to adapt the behaviour.

6 Conclusions

Management and security are closely linked. Access control is essential to protect objects so that only authorized manager agents can perform management operations on them. Security needs to be managed to disseminate relevant policies to the agents that will implement them, specific actions are needed to deal with security violations and flexible policies are needed for response to intrusions. Concepts such as roles are useful both for security and management for grouping the policies applying to a position in an organization.

In this paper we have described approaches to specifying both security and management policy and then discussed the Ponder framework which caters for both types of policy.

There is considerable interest in policy-based systems, from standardization groups, industry and academia. However most of the literature relates to proposed systems or small academic prototypes. There is no reported experience of deploying policy-based management in any large-scale system. It is thus too early to judge whether the promised flexibility of policy based adaptive systems will actually materialize.

Acknowledgements

We gratefully acknowledge the support of EPSRC for research grants GR/L96103 (SecPol) and GR/M86109 (Ponds). We also acknowledge the contribution of our colleagues, Naranker Dulay and Nicodemos Damianou to the ideas expressed in this paper.

References

[1] P. Konstantinos, "Active Networks: Applications, Security, Safety, and Architectures" *IEEE Communications Surveys and Tutorials*, vol. 2, no. 1, 1999, http://www.comsoc.org/livepubs/surveys/index.html.

[2] M. Sloman, E. Lupu, "Policy Specification for Programmable Networks", *Proc. First International Working Conference on Active Networks (IWAN'99)*, Berlin, June 1999, ed. S. Covaci, Springer LNCS, pp. 73-84.

[3] M. Sloman, "Policy Driven Management for Distributed Systems", *Journal of Network and Systems Management*, vol. 2, no.4, 1994, pp. 333-360.

[4] Policy 2001: Policies for Distributed Systems and Networks eds. M. Sloman, J. Lobo. E. Lupu, *Springer Lecture Notes in Distributed Systems Vol. 1995*, Jan. 2001. http://link.springer.de/link/service/series/0558/tocs/t1995.htm

[5] G. Stone, B. Lundy, G. Xie, "Network Policy Languages: A Survey and a New Approach, *IEEE Network*, vol. 15, no. 1, Jan. 2001, pp. 10-21.

[6] http://www-dse.doc.ic.ac.uk/Research/policies/

[7] R.S. Sandhu, E.J. Coyne, H.L. Feinstein and C.E. Youman. "Role-Based Access Control Models", *IEEE Computer*, vol. 29, no.2, 1996, pp. 38-47.

[8] Proceedings of the First ACM/NIST Workshop on Role Based Access Control, Gaithersburg, ACM Press, 1995.

[9] Chen, F. and R.S. Sandhu, “Constraints for Role-Based Access Control”, *Proc. 1st ACM/NIST Role Based Access Control Workshop*, Gaithersburg, USA, 1995.

[10] Blaze M., Feigenbaum J. and Lacy J, “Decentralized Trust Management”, *IEEE Conference on Security and Privacy*, 1996, Oakland, USA, http://www.crypto.com/papers/policymaker.pdf

[11] Blaze M., Feigenbaum J. and Keromytis A. D. “KeyNote: Trust Management for Public-Key Infrastructures”, *Security Protocols International Workshop*, 1998, Cambridge, UK, http://www.cis.upenn.edu/~angelos/Papers/keynote-position.ps.gz

[12] Herzberg, A. Y. Mass, J. Mihaeli, D. Naor and Y. Ravid. “Access Control Meets Public Key Infrastructure, or: Assigning Roles to Strangers”, *IEEE Symposium on Security and Privacy, Oakland,* 2000. www.hrl.il.ibm.com/TrustEstablishment/paper.asp.

[13] S. Jajodia, P. Samarati, and V.S. Subrahmanian, “A Logical Language for Expressing Authorisations”, *IEEE Symposium on Security and Privacy*, Oakland, USA, 1997, pp.31-42

[14] Ortalo, R. “A Flexible Method for Information System Security Policy Specification”, *Proc. 5th European Symposium on Research in Computer Security (ESORICS 98)*. 1998, Louvain-la-Neuve, Belgium, Springer-Verlag.

[15] Corradi, A., R. Montanari, C. Stefanelli, E. Lupu and M. Sloman. “Flexible Access Control for Java Mobile Code”, *16th Annual Computer Security Applications Conference (ACSAC2000)*, Dec 2000, New Orleans USA.

[16] Hashii,B., S. Malabarba, R. Pandey M. Bishop. “Supporting reconfigurable security policies for mobile programs” *Computer Networks*, Elsevier Publishing, vol. 33, June 2000, pp. 77-93.

[17] Lobo, J., R. Bhatia and S. Naqvi. “A Policy Description Language”, *Proc. AAAI*, July 1999. Orlando, Florida, USA, MIT Press, pp. 291-298.

[18] Virmani, A., J. Lobo and M. Kohli. “Netmon: Network Management for the SARAS Softswitch”, *IEEE/IFIP Network Operations and Management Symposium, (NOMS2000)*, ed. J. Hong and R. Weihmayer, Hawaii, May 2000, pp. 803-816.

[19] Distributed Management Task Force, Inc. (DMTF), “Common Information Model (CIM) Specification”, version 2.2, www.dmtf.org/standards/standard_cim.php, June 14, 1999.

[20] Internet Engineering Task Force, Policy Working Group www.ietf.org/html.charters/policy-charter.html

[21] Strassner, J., and E. Ellesson, B. Moore, A. Westerinen, “Policy Core Information Model - Version 1 Specification”, RFC 3060, February 2001, www.ietf.org/rfc/rfc3060.txt

[22] Y. Snir, Y. Ramberg, J. Strassner, “Policy QoS Information Model”, Nov. 2001, www.ietf.org/internet-drafts/draft-ietf-policy-qos-info-model-04.txt

[23] J. Strassner, “Policy and the IETF – Theory and Practice”, Invited presentation, *Policy 2001: Policies for Distributed Systems and Networks*, HP Labs., Bristol, Jan 2001, www-dse.doc.ic.ac.uk/events/policy-2001/

[24] D. Verma, “Policy Based Networking: Architecture and Algorithms”, New-Riders Publishing, 2000.

[25] Enterprise-Viewpoint Reference Model, CD 15414, ISO/IEC JTC1/SC7 N2187, 1999.

[26] Blanc X., Gervais M., Le-Delliou R., “Using the UML Language to Express the ODP Concepts”, *Proc. 3rd. Enterprise Distributed Object Computing Conf.*, Germany, Sep.1999, pp.50-59.

[27] Øyvind Aagedal J, Milošević Z, “ODP Enterprise Language: UML Perspective”, *Proc. 3rd. Enterprise Distributed Object Computing Conf. (EDOC99),* Germany, Sep. 1999, pp.60-71

[28] Linington P, "Options for Expressing ODP Enterprise Communities and Their Policies by Using UML", *Proc. 3rd. Enterprise Distributed Object Computing Conf. (EDOC99)*, Germany, Sep.1999, pp.72-82.

[29] Minsky, N.H. and Pal, P. "Law-Governed Regularities in Object Systems – Part 2: A Concrete Implementation", *Theory and Practice of Object Systems (TAPOS)*, vol. **3**, no. 2, 1997, John Wiley.

[30] Damianou, N., N. Dulay, E. Lupu and M. Sloman, "The Ponder Policy Specification Language", *Policies for Distributed Systems and Networks (Policy2001)*, Springer LNCS 1995, pp.18-38, HP Labs Bristol, 29-31 Jan 2001.

[31] Lupu, E.C. and M.S. Sloman. "Towards a Role Based Framework for Distributed Systems Management", *Journal of Network and Systems Management*, vol. 5, no. 1, 1997, pp. 5-30.

[32] Lupu, E.C. and M. Sloman, "Conflicts in Policy-Based Distributed Systems Management", *IEEE Trans. on Software Engineering*, vol. 25, no. 6, Nov. 1999. pp. 852-869

[33] Dulay N, E. Lupu, M. Sloman, N. Damianou, "A Policy Deployment Model for the Ponder Language", *Proc. IEEE/IFIP International Symposium on Integrated Network Management, (IM'2001)*, Seattle, May 2001, IEEE Press.

[34] R. Darimont, E. Delor, P. Masonet, A van Lamsweerde, "GRAIL/KAOS: An Environment for Goal-Driven Requirements Engineering", *Proc. IEEE 20th Int. Conference on Software Engineering (ICSE'98)*, Vol. 2, pp. 58-62, Kyoto, Japan, April 1998.

[35] M. Bearden, S. Garg, W. Lee, "Integrating Goal specification in Policy-Based Management, *Policies for Distributed Systems and Networks (Policy2001)*, HP Labs Bristol, 29-31 Jan 2001, Springer LNCS 1995, pp.153-17.

Biographies

Professor Morris Sloman is head of the Distributed Software Engineering Section, Department of Computing, Imperial College, London. He has been working on policy based management for 10 years, but his research interests include management, security and design of distributed systems, multimedia systems and mobility. He is editor of a reference book on Management of Network and Distributed Systems (Addison Wesley), and is a member of the editorial board of the Journal of Network and Systems Management. Prof. Sloman is a member of IEEE ComSoc, ACM and BCS. See http://www-dse.doc.ic.ac.uk/~mss for more details and selected papers.

Dr. Emil Lupu (www.doc.ic.ac.uk/~ecl1/) obtained his Diplôme d'Ingénieur from the ENSIMAG, France and his Ph.D. in Computer Science from Imperial College. His research focuses on design, security and mobility issues in large distributed systems with particular emphasis on policy-based network and security management. Dr. Lupu serves on the program committee of numerous conferences and has been program co-chair of the Policy Workshop 2001 and the 5th IEEE Enterprise Distributed Object Computing Conference (EDOC) 2001.