To appear in: Proceedings of the Vth International Symposium on Integrated Network Management IM'97 (formerly knonw as ISINM), San-Diego (U.S.A.), Chapman&Hall, May 1997.

Conflict Analysis for Management Policies

E. Lupu, M. Sloman
Imperial College, Department of Computing,
180 Queen's Gate, London SW7 2BZ, U.K.
E-mail: e.c.lupu@doc.ic.ac.uk, m.sloman@doc.ic.ac.uk

Abstract

Policies are a means of influencing management behaviour within a distributed system, without coding the behaviour into the managers. **Authorisation** policies specify what activities a manager is permitted or forbidden to do to a set of target objects and **obligation** policies specify what activities a manager must or must not do to a set of target objects. Conflicts can arise in the set of policies. For example an obligation policy may define an activity which is forbidden by a negative authorisation policy; there may be two authorisation policies which permit and forbid an activity or two policies permitting the same manager to sign cheques and approve payments may conflict with an external principle of separation of duties. This paper reviews the policy conflicts which may arise in a large-scale distributed system and describes a conflict analysis tool which forms part of a Role Based Management framework. Management policies are specified with regard to domains of objects and conflicts potentially arise when there are overlaps between domains. It is not desirable or possible to prevent overlaps and they do not always result in conflicts. We discuss the various techniques which can be used to determine which conflicts are important and so should be indicated to the user and which potential conflicts should be ignored because of precedence relationships between the policies. This reduces the set of potential conflicts that a user would have to resolve and avoids undesired changes of the policy specification or domain membership.

Keywords

Distributed systems management, management roles, management policies, conflict detection, conflict resolution, policy precedence.

1 INTRODUCTION

There has been considerable interest recently in policy based management for distributed systems (Sloman, 1994; DSOM, 1994; Magee, 1996; Koch, 1996). Separating policies from the managers which interpret them permits the modification of the policies to change the behaviour and strategy of the management system without recoding the managers. The management system can adapt to changing requirements by disabling policies or replacing old policies with new ones without shutting down the system. We are concerned with two types of policies: **authorisation** policies which specify what activities a subject is permitted or forbidden to do to a set of target objects and **obligation** policies which specify what activities a subject must or must not do to a set of target objects. The subject or target of a policy is usually expressed as a domain of objects and applies to all objects in the domain so a single policy can be specified for a group of objects. This helps to cater for large scale systems in that it is not necessary to define separate policies for individual

objects in the system, but rather for groups of objects. We permit the specification of both positive and negative authorisation policies and require explicit authorisations i.e. non authorised invocations are forbidden.

In a large distributed system there will be multiple human managers specifying policies which are stored on distributed policy servers. Policy inconsistencies can arise due to omissions, errors or conflicting requirements of the managers specifying the policies. For example an obligation policy may define an activity a manager must perform but there is no authorisation policy to permit the manager to perform the activity. Conflicts can also arise between positive and negative policies applying to the same objects. In general, whenever multiple policies apply to an object there is a potential for some form of conflict but it is essential that multiple policies should apply in order to cover the diversity of management functions and of management domains. For example there may be different policies relating to security, monitoring, or configuration which apply to a set of objects reflecting different management functions which may be performed on the objects. Similarly the policies specified for the network, sub-network and workstation domains will all propagate to the network objects inside the workstation.

In this paper we describe the tools we have developed for analysing policy specifications to determine inconsistencies and conflicts. We use **roles** as the means of grouping policies related to a particular manager position and then managers can be assigned or removed from the position without changing the policies (Lupu, 1997). We also define the relationships between roles with regard to the use of shared resources or with regard to the organisational structure e.g. the manager assigned to role A has the right to assign a task to the manager assigned to role B. A large scale distributed system will have very large numbers of objects and policies distributed around the system, so the conflict detection cannot be centralised but also has to be distributed. Our use of roles and inter-role relationships provides a scope for the conflict detection and helps to limit the number of policies which have to be examined in order to determine conflicts. We assume more specific policies take precedence over less specific ones in order to automatically resolve some conflicts and so reduce the number that human administrators have to resolve. This paper focuses on techniques and tool support for off-line conflict detection and resolution, although some conflicts can be detected only at run time.

In section 2 of the paper we give more details of the domains, policies and roles which form our management framework. Section 3 discusses the types of policy conflicts we need to detect. In section 4 we explain our approach to conflict detection, policy precedence relationship and describe the tools we have implemented.

2 MANAGEMENT FRAMEWORK

The main components of our management framework are domains for grouping objects, a policy service to support the specification and storage of policies and roles to reflect the organisational structure, responsibilities and relationships between management positions.

2.1 Domains

Domains provide a flexible means of partitioning the objects in a large system according to geographical boundaries, object type, management functionality, responsibility, and authority or for the convenience of human managers (Sloman, 1994a & b). A domain groups the management interfaces of objects and may include other domains (which are called subdomains of the parent domain). An object or subdomain may be a member of multiple parent domains.

The **Domain Browser** is a tool for navigation in the domain structure. In Figure 1 the current domain /Example/Org1/Policies contains two policy objects, has one subdomain (SharedPolicies) and is a member of two parent domains (AllPolicies and Org1).

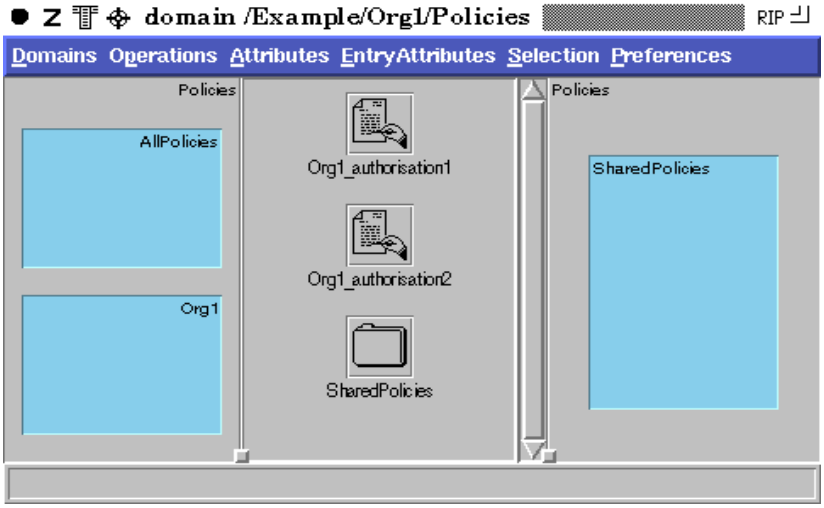


Figure 1 The domain browser.

2.2 Policy Service

In this section we give some examples of obligation and authorisation policies and an overview of the notation used.

Authorisation policies define what activities a subject (manager or agent) can perform on a set of target objects or what monitored information can be received e.g.

> **A+** *Sregion_agents {"lu1", "lu2": enable(); disable(); reset(); off()} *Sregion
> **when** (time > 08:00) && (time < 18:00)
>
> Subjects in the Sregion_agents domain are permitted to perform enable, disable, reset or off operations on objects of type lu1 and lu2 (line units) in the Sregion domain, between hours 08:00 and 18:00.

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. Constraints can be specified to limit the applicability of the policy based on time or attributes of the objects to which the policy refers.

> **O+** **on** overload_event *Sregion_agent {"lu1": disable(), "lu2": enable()} *Sregion
> **when** (08:00 < time) && (time < 18:00)
>
> This positive obligation policy is triggered by an overload event and results in the agent disabling line units of type lu1 and enabling line units of type lu2.
>
> **O–** x:*Sregion_agent {"lineunit": enable(); disable(); reset(); off()} *Sregion
> **when** x.state == standby
>
> This negative obligation policy specifies that standby agents must not perform control operations on line unit objects even though they may be authorised to do so.

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

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

The **mode** of the policy distinguishes between positive obligations (**O+**), negative obligations (**O-**), positive authorisations (**A+**) and negative authorisations (**A-**). Negative obligations should be read as "obliged not to" and can be considered as 'filters' (Moffett, 1993) to prevent the actions specified in positive obligation policies being performed under certain circumstances, which is why they cannot be triggered.

The **subject** of a policy specifies the human or automated managers and agents to which the policies apply and which interpret obligation policies. The **target** of a policy 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. Both subject and target can be defined using a domain scope expression which identifies a set of objects in terms of union, difference, intersection and membership operators over sets of domains and objects. By default, policies propagate to subdomains within a domain and hence to indirect members of the parent domain, but the scope expression can limit this propagation to direct members. An advantage of specifying policy scope in terms of domains is that objects can be added and removed from domains to which policies apply without having to change the policies. The domain scope expressions are evaluated when detecting potential conflicts to determine the subject and target sets to which the policy applies. The **actions** specify what must be performed for obligations and what is permitted for authorisations. It consists of method invocations or a comment and may list different methods for different object types. Multiple actions can also be specified. The **constraint** limits the applicability of a policy, e.g. to a particular time period, or making it valid after a particular date. 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 manual cross reference list of policies may be kept e.g. to refer to the authorisation policy granting permission for an obligation policy's activity.

The policy service provides tool support for defining policies and disseminating polices to the relevant agents which will interpret them. It also permits policies to be enabled, disabled or removed from the agents (Marriott, 1996a & b). Policies are implemented as objects which can be members of domains (see Figure 1) so that authorisation policies can be used to control access to the policies stored in a policy server, e.g. to permit only authorised managers to define and modify policies.

2.3 Roles

Specifying organisational policies for human managers in terms of a **manager position** rather than the person permits the assignment of a new person to the manager position without respecifying the policies referring to the duties and authorisations of that position. The tasks and responsibilities corresponding to the position are grouped into a **role** associated with the position (which is essentially a static concept in the organisation). These definitions correspond to the concepts of classic Role Theory which postulates that individuals occupy positions inside an organisation and associated with the position are a set of activities (including the required interactions) that constitute the role of that position (Biddle, 1979).

Manager positions can be represented as domains and we can consider a **role** to be a set of management policies relating to a particular subject i.e. the Manager Position Domain (Sloman 1994a). A manager may be assigned to a role by including his **User Representation Domain (URD)** in the manager position domain, and the policies of the role will propagate to the URD and objects contained in it. The URD is a persistent representation of the manager in the system. The problem with this approach is that when a manager is assigned to multiple roles all the policies propagate to the URD, so the roles cannot be distinguished and the manager could perform a task in one role with the rights from another. An alternative way of assigning a manager to a role is by specifying a policy authorising him to create an agent in the manager position domain. The manager

assigned to more than one role interacts with an adapter object in his URD which forwards the invocations to the relevant agent in the position domains. The adapter would provide a separate context for each role and thus permit the manager to keep activities pertaining to each role distinct. In this respect the adapter is similar to an X server maintaining different windows with different active shells Figure 2. Note that although logically placed in the position domains the agents may be implemented as threads of the adapter object to improve performance.

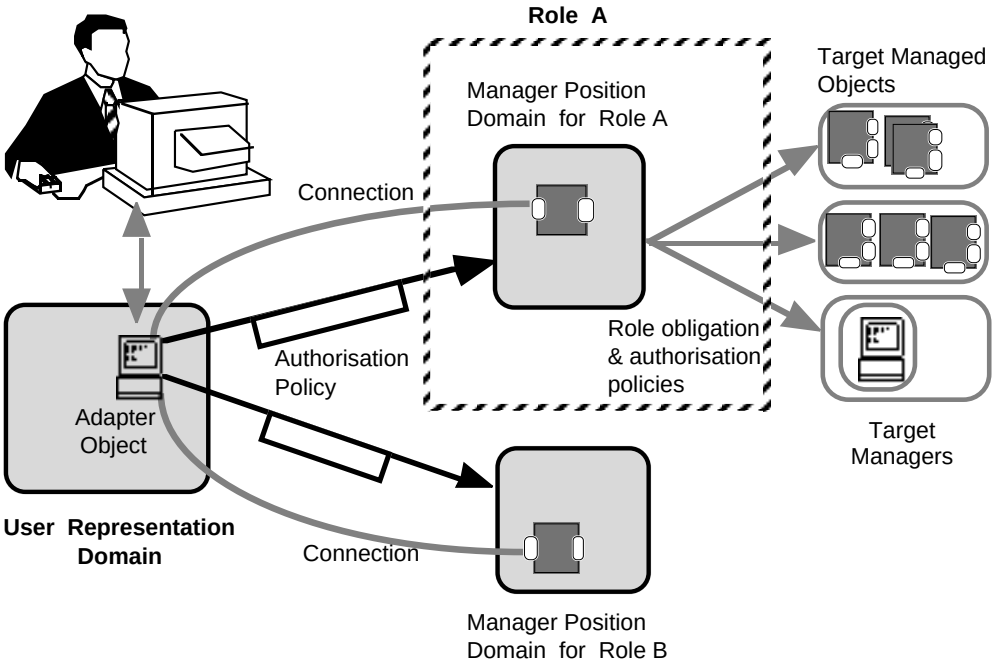


Figure 2 Management Roles.

There is a need for interactions between roles (e.g. delegating a task from one role to another or coordinating access to shared objects). Our management framework therefore caters for specification of role relationships using policies, interaction protocols and concurrency constraints. Figure 3 represents the extended role model as presented in (Lupu, 1997). The policies within a role or between related roles provide a scope within which to search for conflicts.

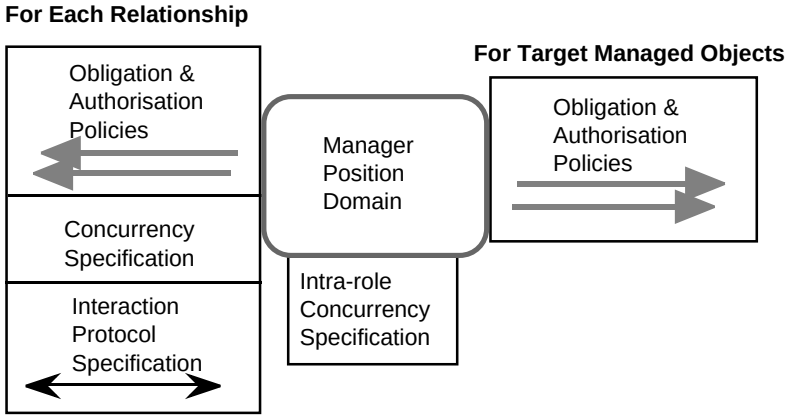


Figure 3 The extended role model.

3 CONFLICT CLASSIFICATION

Modality conflicts are inconsistencies in the policy specification which may arise when two or more policies with modalities of opposite sign refer to the same subjects, actions and targets. This occurs when there is a triple overlap between the sets of subjects, targets and actions as shown in Figure 4, and so can be determined by syntactic analysis of polices. There are three types of modality conflicts:

- **O+/O-** the subjects are both required and required not to perform the same actions on the target objects.
- **A+/A-** the subjects are both authorised and forbidden to perform the actions on the target objects.
- **O+/A-** the subjects are required but forbidden to perform the actions on the target objects (obligation does not imply authorisation in our case).

A second type of conflict refers to the consistency between what is contained in the policies i.e. which subjects, targets and actions are involved and external criteria such as limited resources or the overall policies of the organisation. An example of this type of conflicts arises from the principle of separation of duties (Clark, 1987) e.g. the same manager cannot authorise payments and sign the payment cheques. These conflicts are **application specific** and cannot be determined directly from the policy specifications – additional information is needed to specify the conditions which result in conflict. These can be specified as a **meta-policy** i.e. a policy about permitted policies. Several types of application specific conflicts such as: conflict of priorities for resources, conflict of duties, conflict of interests, multiple managers conflict and self-management conflict have been identified in (Moffett, 1994) and classified according to the overlaps between the subject, action and target sets.

Modality conflicts arise from overlapping domains but it is impractical to prevent these overlaps (see 4.1 a) as there is a need for multiple policies to apply to a domain to reflect partitioned responsibility and the various different management functions that can be performed on target objects e.g. different managers may be responsible for maintenance and security relating to a domain of workstations. In the following, we discuss the precedence relationships which can help to resolve modality conflicts then describe our approach to specifying meta-policies to detect application specific conflicts.

4 CONFLICT DETECTION

Conflict detection between management policies can be performed statically for a set of policies in a policy server or at run time. A run-time mechanism acts as a filter preventing activities that must not be performed (O-) or are not permitted (A-) (Moffett, 1993). The advantage is that all the constraints of the policies can be evaluated at run time and so all conflicts can be detected, but some conflicts may really be specification errors and should rather be detected by static analysis c.f. compile time vs. run-time error detection for programming languages. The disadvantages of static analysis are that policy constraints cannot be evaluated, as they depend on run-time state, and domain membership may change, so only potential rather than actual conflicts can be detected. Both static and run-time conflict detection are needed, but this paper concentrates on a static conflict detection tool which assists the users specifying policies, roles and relationships. In the following we discuss some principles for the detection of the modality conflicts and present an implementation of the conflict detection tool.

4.1 Policy Precedence Relationships

As previously mentioned, modality conflicts result from a triple overlap between the subjects, actions and targets of the policies. In a typical organisation there will be some

general policies pertaining to all staff in the organisation as well as more specific policies relating to staff in a department or section. Staff may also be members of many different domains. Detecting the triple overlaps between policies with modalities of opposite signs would therefore detect many potential conflicts which do not result in actual conflicts. Using a policy precedence relationship can substantially reduce the number of conflicts indicated to the user and permit apparently inconsistent specifications. There are several principles, outlined below, for establishing this precedence. The choice between them has to be guided by which conflicts should be ignored and how easy it is for the human user to understand the decisions and selection of the conflict detection tool using this principle i.e. how intuitive the principle is.

a) Negative policies always have priority

It is quite common for negative authorisation policies to always override positive ones so that a forbidden action will never be permitted. Consider the following policies:

/ All users are forbidden to access the system files */*

P1 A- @/users { reboot() } @/workstations

/ The system administrators are authorised to reboot the workstations */*

P2 A+ @/users/sys_admin { reboot() } @/workstations

Policy P1, being negative has priority over P2 so the system administrators are denied access to the system files, but then they cannot perform their function. To resolve this conflict it is necessary either to rewrite policy P1 or to exclude the system administrators from the /users domain. Although our access control system assumes a negative default authorisation policy and so we would not specify P1 but only P2. Database security systems often implement negative authorisation so we permit it at the specification level.

b) Assigning explicit priorities

A user can assign explicit priority values to policies to define a precedence ordering, 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.

c) Distance between a policy and the managed objects

The concept of calculating the *distance* between a rule (policy) and the objects it refers to has been introduced in (Larrondo-Petrie, 1990) for authorisation policies in an object oriented database. Priority is given to the policy applying to the *closer* class in the inheritance hierarchy when evaluating access to an object referenced in a query. For example the access policy applying to a foreign student is the one applying to a student and overrides the general access policy applying to a person if foreign student is a subclass of student which is a subclass of person. The distance between the policy and the objects to which it applies indicates the relevance of the policy and can be precisely evaluated from the number of levels of refinement of the organisational policies. In the general there is a compromise between the complexity and the intuitiveness of the distance to be evaluated. A distance which is intuitive may not correctly evaluate the importance of a policy in all the cases and a complex calculated distance may not be intuitive enough for the human user to understand the selection and priorities assigned to a policy during the conflict detection process e.g. the priority could be based on the product (refinement level) * (last modification date).

d) Specificity related to domain nesting

The principle here is that a more specific policy i.e. a policy applying to a subdomain refers to fewer objects so overrides more general policies applying to an ancestor domain. This concept has been introduced in Miró (Heydon 90) and is a particular case of the previous concept of distance. Considering the specificity of a policy with regards to the

objects it applies to is an intuitive concept in a domain based system. A subdomain of objects is created for a specific management purpose – to specify a policy that differs from those applying to the objects in the parent domain. For example the system administrators are a particular group of users which have access to the system files despite the general policy denying access to all users of the system. The other policies applying to all the users apply then to the system administrators in the same way. Precedence based on domain nesting can thus be used to allow conflicting specifications by automatically resolving some conflicts.

In section 4.2 we describe how domain nesting can be used within conflict detection to reduce the number of potential conflicts. We recognise that this principle does not apply successfully to all the situations i.e. there are cases in which it is desirable that a global policy overrides more specific ones. For this purpose the conflict detection can be performed with precedence relationships optionally disabled. The following two sections examine the importance of the overlaps between domains while applying the domain nesting principle and indicates the cases where inconsistencies still remain.

4.2 The importance of overlaps in modality conflicts

The analysis for conflicts of a set of policies enumerates all the subject, action target tuples which have a different set of policies applying to them. This makes it easier to determine where a conflict occurs and where precedence resolves a potential conflict.

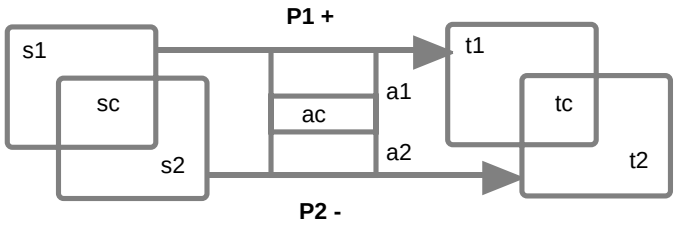


Figure 4 Overlapping Subjects, Targets and Actions.

Consider the policies **P1** and **P2** represented in Figure 4 with P1 being positive and P2 being negative. Let us call the overlapping areas s_C, a_C and t_C for common subjects, actions and targets. The triple overlap between the policies P1 and P2 creates three tuples to which different sets of policies apply:

- P1 applies to $<s_1\text{-}s_C, a_1\text{-}a_C, t_1\text{-}t_C>$
- P2 applies to $<s_2\text{-}s_C, a_2\text{-}a_C, t_2\text{-}t_C>$
- P1 and P2 apply to $<s_C, a_C, t_C>$

Neither P1 nor P2 is more specific so a conflict is indicated to the user as their modalities are of opposite sign. Now consider a policy P3 (shown in Figure 5) defined by the tuple $<s_3, a_3, t_3>$ such that $s_3=s_C$, $a_3=a_C$ and t_C is a subset of t_3 which is a subdomain of t_2.

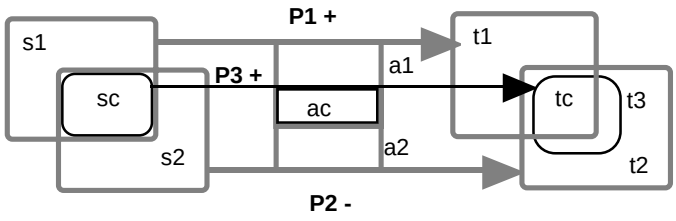


Figure 5 Adding a policy.

We now have the following tuples and policies:

- P1 applies to <s1-sc, a1-ac, t1-tc>
- P2 applies to <s_2-s_C, a_2-a_C, t_2-t_3>
- P1, P2 and P3 apply to <s_C, a_C, t_C>
- P2 and P3 apply to <sc, ac, t3-tc>

P3 is positive and is more specific than P2 so it overrides P2 in the areas where they overlap i.e. for the tuple <s_C, a_C, t_3-t_C> and <s_C, a_C, t_C>. Since P1 and P3 have the same modality, no conflicts would be indicated.

Note that when displaying the result of a conflict detection check it is important to provide the user with the information regarding which policies conflict, where precedence overrides conflicts and to which tuples <subjects, actions, targets> these policies apply.

4.3 Limitations of domain nesting based precedence

The domain nesting precedence determines all policies which apply to a tuple of subjects actions and targets and gives precedence to policies which apply to a more specific set of subjects, targets or both. There are cases in which precedence cannot be established because the sets are equal, the subject sets are more specific but the target sets are less

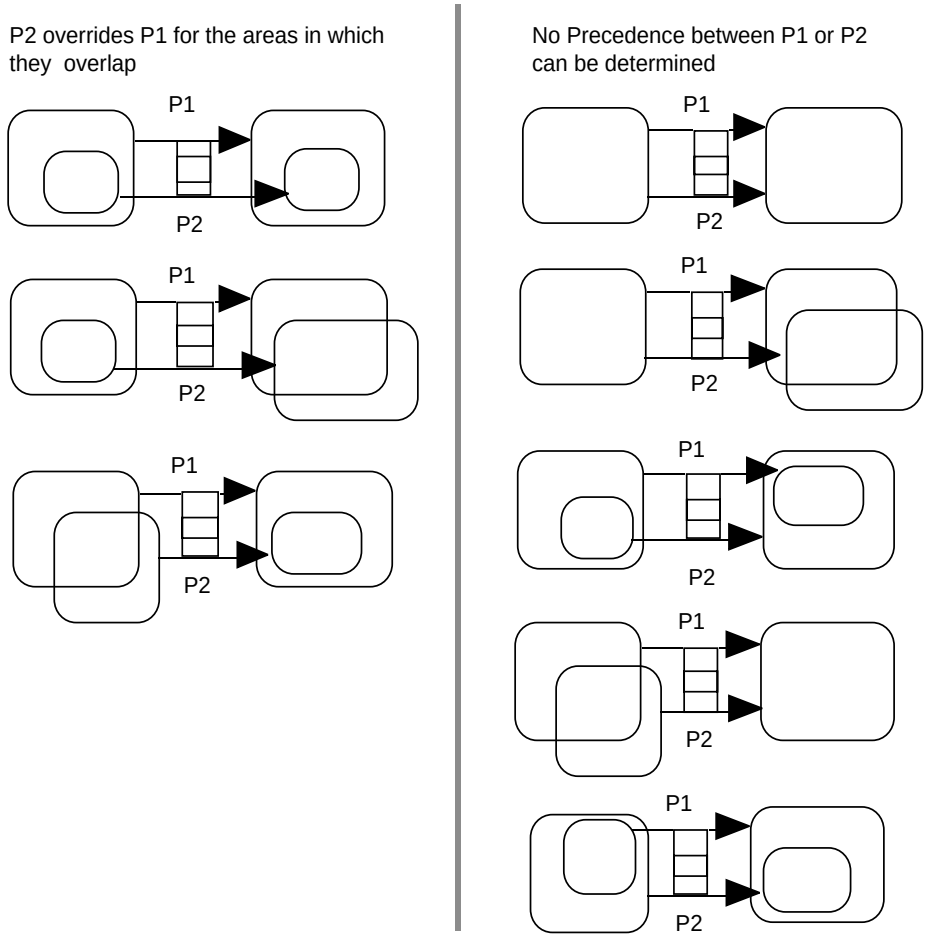


Figure 6 Precedence between policies.

specific or vice versa. The various situations where precedence can or cannot be established between two policies are shown in Figure 6. Note that that precedence may based on a policy's subject *or* target set so it is not an ordering relation because it is not transitive. There is no precedence relationships between obligations and authorisations since an obligation overriding an authorisation would convey the implicit assumption that the obligation implies authorisation and this is not true for our policies.

4.4 A conflict detection tool

Implementation issues

The conflict detection tool detects overlaps between policies and applies domain nesting based precedence. The domains and policies are distributed among several servers so Corba invocations are used for retrieving the policies and querying domains to evaluate their sets of subjects, actions, and targets. In theory, all policies in the system need to be checked for overlaps but this is impractical. Instead, we permit the user to specify the scope of policies to be checked, for example the policies applying to particular roles or the policies of a relationship between roles. Policies or domains of policies can be dragged from the domain browser window (Figure 1) into the conflict detection window to establish the set of policies over which the conflicts are to be detected. The meta policies discussed in section 4.5 can also explicitly define the scope to which they apply.

Since there are cases in which a more specific policy should not take precedence, domain nesting precedence can be optionally disabled so all the overlaps are indicated. Finally an analysis option also permits all the tuples of subjects, actions and targets and the policies applying to them to be displayed even if there are no conflicts as it is useful to examine which policies apply to which tuples. When enabled, the precedence relationship between policies is indicated by arrows between the policy icons as shown in Figure 7.

Example

Policies are implemented as objects in the system. This example shows the use of the conflict detection tool while specifying policies for managing other policy objects. Consider the case of three organisation domains Org1, Org2, Org3 sharing the same computer system. Each organisation has two managers Oi_m1 and Oi_m2 {i ∈ 1..3} and a set of policies which includes a domain of shared policies about the general use of the computer system (see Figure 1). Each of the organisations wants to retain control of all its policies including the shared policies. In particular each organisation has two policies stating that its managers can perform various operations on all its policies and that the managers from the other organisations are prohibited from performing the operations retract(), disable() or delete() on any of the objects contained in the organisation's policy domain. These policies have the following format (only shown for Org1):

Org1_authorisation1 **A+** @/Org1/Managers {create(); delete(); distribute(); enable(); disable(); retract()} @/Org1/Policies

Org1_authorisation2 **A-** @/Org2/Managers + @/Org3/Managers {delete(); disable(); retract()} @/Org1/Policies

Note the '@' symbol selects all non domain objects in nested domains. (With a default negative authorisation, the second policy could actually be revised to only permit create and enable to give the same effect but we will ignore that for the purposes of this example.) The managers of one organisation are subjects of three policies: one authorising all the operations on the objects of the Policies domain in their organisation and two others prohibiting some operations on the policy objects in the policy domain of the other two organisations. A potential conflict arises from the presence of the shared Policies subdomain in each organisation. Since the positive authorisation policy is more specific than the two others (it relates to the managers of Org *i* while the others relate to the

managers of Org *x* + Org *y*) no conflicts are detected because the positive authorisation policies override the negative ones, as shown in the conflict detection window of Figure 7.

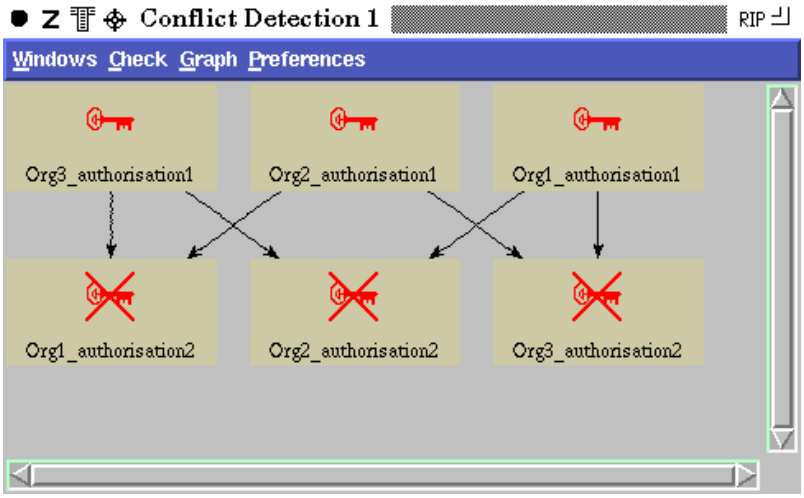
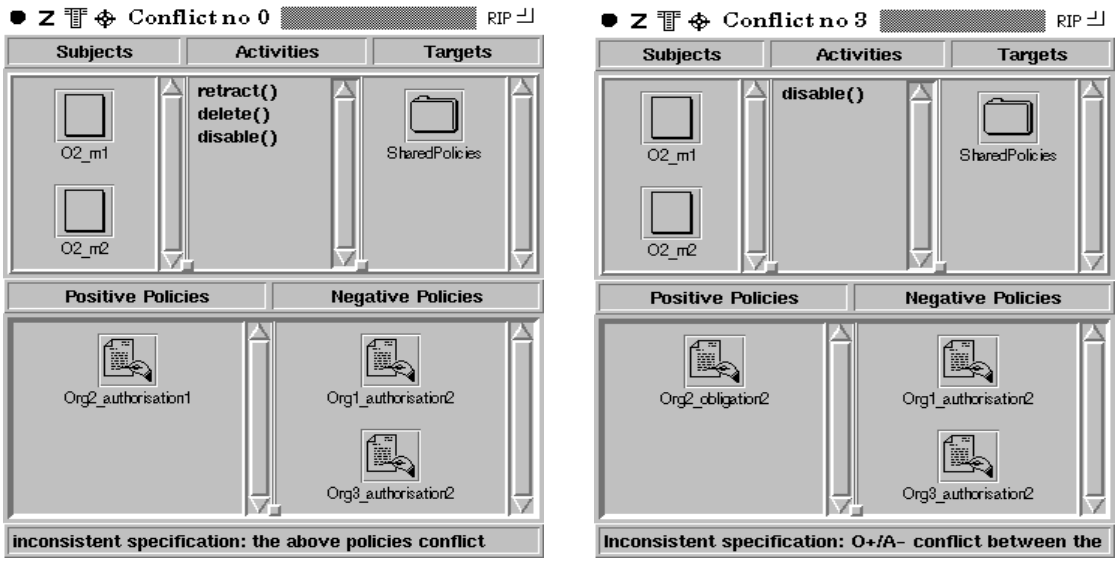


Figure 7 The conflict detection window. Shows positive authorisation (keys) overriding negative authorisation (crossed out keys).

If the check for conflicts is performed without the domain nesting based precedence, conflicts such as the one shown in Figure 8a are detected. The subjects, actions, targets tuple is shown in the upper part and the conflicting policies to which no precedence applies are shown in the lower part of the screen. Similarly remaining conflicts can be displayed after applying domain nesting precedence, although there are none in this example. Policy icons can be dragged from this window onto a policy editor window for viewing and revising if required.



(a) **(b)**

Figure 8 Examples of detected conflicts.

Consider a policy refined from a more abstract obligation policy specifying that the managers from **Org2** must (modality **O+**) disable the policies on the failure of the policy server.

Org2_obligation1 **O+ on** maps_failure @/Org2/Managers { disable() } @/Org2/Policies

If domain nesting precedence is used, the access should be granted since the positive authorisation takes precedence over the negative ones. With precedence disabled, the **O+/A-** conflict is also detected as shown in Figure 8b.

4.5 Meta-Policies

Meta-policies specify application specific consistency constraints pertaining to the contents of policies. Meta-policies constrain the set of acceptable policies in terms of their attributes. They can be expressed as logical predicates applying to the sets of policy objects determined by a domain scope expression (dse). For example a conflict for resources may arise when the number of objects in the target domain of any two policies is greater than 11.

$$\forall P1, P2 \in < dse >$$

$$fail \leftarrow P1.t\,arg\,ets + P2.t\,arg\,ets > 11$$

The solution of the following Prolog code gives all the conflicting policies.

```
checkResourcesNumber(P, Q, Res) :-
    numberOfTargets(P, N1), numberOfTargets(Q, N2), Res = N1 + N2.
checkRes(P, T) :-  checkResourcesNumber(P, T, Res), P \= T, Res > 11.
check1(Bag) :- findall([P, T], checkRes(P, T), Bag).
```

We have been experimenting with meta-policies by implementing the predicate specification in Prolog for the cases presented in (Moffett, 1994). The policies contained in the Conflict Detection window are automatically translated into Prolog assertions. A Prolog process can then be started from the conflict detection tool loading the file containing the translated policies. The predicate specifying the conflict is then a query on the assertions database which gives the policies in conflict.

5 RELATED WORK

Our concept of domain nesting precedence is based on that of Miró (Heydon, 1990), but they only deal with authorisation policy for file system security. Sandhu (1996) presents constraints which are similar to our meta-policies, but the notation used is not described. The work presented in (Michael, 1993) relates to general policies, expressed in natural language and modelled in an Entity Relationship representation. A theorem prover is used to detect the inconsistencies. The "law governed systems" of (Minsky, 1996) implements a common global set of constraints by means of filters in every node which check that all interactions are consistent with the global law.

Another approach, used to detect feature interaction in telecommunication systems (Griffeth, 1993), considers policies as goals and applies planning techniques to resolve situations with incompatible goals. Planning techniques for conflict management are also used in Distributed Artificial Intelligence (Lander, 1994). In the case of our management policies such techniques could be used only in conjunction with the refinement of the policies. Koch (1996) uses a policy notation based on ours and establishes a semantic graph model to detect ill-behaved policy sets with unsatisfiable pre-conditions. This can also be used to perform "what-if" analysis on chains of policies prior to execution.

Deontic Logic provides the closest approximation of our management policies in the context of a logic system. A model of policies as deontic logic statements for office automation can be found in (Ong, 1993). However Standard Deontic Logic also relies on the axiom of inter-definability which defines a permission as **P** = ¬ **O**¬ **P**. No such assumption is made between our authorisation and obligation policies. However a number

of new logical systems with slightly different axioms are emerging and this may be of interest for our policies.

6 CONCLUSIONS

This paper has presented the integration of a conflict detection tool in a more general role and policy based framework for distributed systems management. We perform off-line, static analysis of a set to policies to determine two types of conflicts: (i) modality conflicts which can be checked by analysing the syntax of the policies and (ii) application specific conflicts with external constraints which we express as meta policies. Modality conflicts arise from a triple overlap between the subjects, actions and targets of the policies, but it is not practical nor desirable to prevent these overlaps. We make use of a precedence relationship based on the specificity of the policies with respect to domain nesting to reduce the number of potential overlaps indicated to a user, as we consider this to be an effective and intuitive precedence relationship. Roles are an important management concept but also provide a scope to limit the set of policies to be analysed.

Another aspect of policy analysis relates to determining the policies applying to a particular subject or target. Our policies explicitly identify both subject and target and the domain service maintains the list of policies applying to a domain so this is comparatively easy to do, but has not yet been implemented.

We have implemented a prototype role framework which supports distributed policy and domain servers and analysis of a set of policies, indicating conflicts as well as precedence relationships. This will enable us to experiment in realistic situations and evaluate the use of the precedence relationship. Our approach is to detect as many conflicts as possible at specification time, rather than leaving them to be detected at runtime. The user can then modify the policies to remove conflicts. This has been implemented using a Corba based distributed programming environment.

Further work remains to be done on the use of dynamic run-time conflict detection within policy interpreters and what to do about conflicts which have been detected. Our meta policy specification language also need further refinement, as translating all policy specifications into Prolog assertions is a rather “heavy handed” approach.

7 ACKNOWLEDGEMENTS

We gratefully acknowledge financial support for the EPSRC RoleMan project (GR/K 37512) and British Telecom for the Management of Multimedia Networks project. We are grateful to Jonathan Moffett for many invaluable comments which have improved this article. We acknowledge the contribution of our colleagues to the concepts described in this paper – in particular Nicholas Yialelis and Damian Marriott.

8 REFERENCES

Biddle, B. and Thomas, E. Eds. (1979) *Role Theory: Concepts and Research*. New York, Robert E. Krieger Publishing Company.

Clark, D. and Wilson, D. (1987) A comparison of Commercial and Military computer security Policies. *IEEE Symposium on Security and Privacy*.

DSOM (1994) Proceedings of the IEEE/IFIP Distributed Systems Operations and Management Workshop, Toulouse (France).

Griffeth, N. and Velthuijsen, H. (1993) Reasoning about goals to resolve conflicts. *Int. Conf. on Intelligent Cooperative Information Systems*, Los Alamitos (Calif.), IEEE Computer Society Press, 197–204

Heydon, A. et al. (1990) Miró: Visual Specification of Security. *IEEE Transactions on Software Engineering*, **16**(10), 1185-1197.

Koch, T. et al. (1996). Policy Definition Language for Automated Management of Distributed System. *IEEE 2nd. Int. Workshop on Systems Management,* Toronto (Canada).

Lander, S. E. (1994). Distributed Search and Conflict Management Among Reusable Heterogeneous Agents. Ph.D. Dissertation, University of Massachusetts, Amherst, (USA).

Larrondo-Petrie, M. et al. (1990) Security Policies in Object-Oriented Databases. *IFIP Database Security, III: Status and Prospects*, Elsevier Science Publishers B.V. (North-Holland).

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

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

Marriott, D. and Sloman M. (1996a). Management Policy Service for Distributed Systems. *Proc. IEEE Third International Workshop on Services in Distributed and Networked Environments (SDNE 96)*, Macau, 2–9.

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

Michael, J. (1993) A Formal Process for Testing Consistency of Composed Security Policies. Ph.D. Dissertation, George Mason University, Fairfax, Virginia.

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

Moffett, J. et al. (1993) The Policy Obstacle Course: A Framework for Policies Embedded within Distributed Computer Systems. Technical Report, Schema/York/93/1, Department of Computer Science, University of York (UK).

Moffett, J. and Sloman M. (1994) Policy Conflict Analysis in Distributed System Management. *Ablex Publishing Journal of Organisational Computing*, **4**(1), 1–22.

Ong, K. L. and Lee, R. M. (1993). A Logic Model for Maintaining Consistency of Bureaucratic Policies. *26th Annual Hawaii International Conference on System Sciences*, Hawaii, IEEE Computer Society Press. Vol. III, 503–512

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

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

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