INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Diplomarbeit

An Examination of Requirements for Metapolicies in Policy-Based Management

Bearbeiter:	Marco Avitabile
Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Boris Gruschke Stephen Heilbronner Helmut Reiser

INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Diplomarbeit

An Examination of Requirements for Metapolicies in Policy-Based Management

Bearbeiter:	Marco Avitabile
Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Boris Gruschke Stephen Heilbronner Helmut Reiser

Abgabetermin: 15. November 1998

Hiermit versichere ich, daß ich die vorliegende Diplomarbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, 15. November 1998

....

(Unterschrift des Kandidaten)

Abstract

Policy-based management is seen as an advantageous concept for narrowing down the distance of high-level management objectives to enforceable management actions. The application of policy-based management entails a demand for managing the associated activities in their entirety. *Metapolicies* have been suggested for this purpose. It needs to be investigated to what degree they can play a useful part in this process.

This thesis presents an overview of how a management policy for an information system goes through different stages of a life cycle. Through the presentation of a developed enhanced policy life cycle, the complexity of the necessary processes for policy-based management is exposed, leading to the need for using *metapolicies* to assist policy management. In addition to the normal understanding that metapolicies are “policies about policies”, the concept that metapolicies are also “policies governing the policy life cycle” is introduced. The management aspects for areas affected by the policy life cycle are described and classified, enabling a clearer understanding of the significance of metapolicies. This approach differs from previous related work in that it highlights how the developed concept is driven by system management, and how metapolicy can eventually become an underpinning concept of a complete management system.

In addition to this conceptual work, a design to integrate metapolicies into an existing policy-based management system is given focusing on reusing existing infrastructure. Suggestions to put the metapolicy concept more completely into action are presented.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In recent years, the management of information technology systems has been found to be increasingly based on the use of policies. This is due to the fact that policy-based management is very flexible in the sense that the use of policies can be separated from any particular application. In addition, policy-based management entails a different approach to performing management tasks since it specifies management at different levels of abstraction.

The integrated application of policy-based management is achieved by the cooperation of interconnected processes. These processes need to be managed in a way which supports the policy concept.

In this context, *metapolicies* are an emerging tool for managing policies. Metapolicies have been introduced in the literature as "policies about policies". Using them for managing policy related processes, however, is a new approach.

1.2 Objective and Contribution

The objective of this thesis was to investigate whether there is a need for metapolicies and which tasks in a policy-based management they can carry out.

As a result of my investigation into the policy moving processes, I introduced an enhanced policy life cycle. Then, I derived a demand for managing the policy life cycle and introduced a metapolicy concept to accomplish this task. Finally, I developed a design for integrating some basic metapolicy support into an existing policy-based management application.

1.3 Approach

I chose to employ the policy approach for policy management allowing the reuse of existing infrastructure. Metapolicies for management serve as the basis for management with policies.

1.4 Outline

In chapter 2, a survey of common policy approaches is presented introducing the policy concept and its terminology for information systems management.

Chapter 3 takes a critical look at the understanding of metapolicies in the literature.

In the next chapter, an enhanced policy life cycle is developed clarifying the states a policy has to go through. Also in chapter 4, the supporting policy processes are characterised.

On the basis of this, chapter 5 introduces the concept of metapolicies for the management of the policy life cycle and its related processes. A number of examples show how metapolicies can be used for policy management.

Chapter 6 presents a design I developed to integrate metapolicies into an existing policy-based management system. It focuses on reusing existing infrastructure. Suggestions to put the metapolicy concept more completely into action are presented.

Chapter 7 summarises the work and presents an brief outlook.

Chapter 2

Survey of Management Policy Approaches

In this chapter, the concept of an information systems management policy is introduced by presenting an overview of common management policy approaches.

2.1 Management Policies

A variety of management policy definitions exist due to different viewpoints and different intentions. Common to all, the following definition can be given [Koch 97]:

> *A management policy defines the desired behaviour of a system or of several system components.*

There are a number of reasons why management policies are advantageous for the specification of the desired behaviour. Most of them are a result of the properties of policies:

- Management policies provide for the possibility of centralising knowledge. They are defined at a single or just a few places. This reduces the danger, that similar situations might be interpreted differently by managers reacting differently to the same problem.

- The policy definition extracts management knowledge from experts. This knowledge can be reused. Repeatedly solving the same management problems can be avoided.

- The policy concept is very expressive and can be used for a variety of different management areas.

- As opposed to a reactive management approach, policies aim at describing operational strategies.
- A common format for describing management information in many areas creates a potential for higher rate of automation.
- Policies can support narrowing down the gap between high-level management objectives and enforceable management actions.

The last item is one of the strengths of the policy approach although it is not easy to accomplish. *Refinement* is necessary to create a set of enforceable management actions from a high-level policy (objective). By refinement, a policy is bound to the actual information systems environment and a *policy hierarchy* is created.

In the following sections, different policy approaches are described giving a more complete definition and understanding of management policies.

2.2 Policy Work at the Imperial College

One of the most important contributions to the area of policy-based management is the work of Prof. Moris Sloman and affiliated researchers at the Imperial College, London.

The policy approach developed at the Imperial College has been widely accepted. Therefore, the terminology related to policies very often follows the one introduced in their papers. Unless otherwise noted, this thesis will also use the same terminology.

2.2.1 Initial Research Interest on Policies

The research interest in policies started early with the problem of access control to resources. In [MoSl 88] policies are presented to define rules which regulate how people and programs acting on their behalf can access resources in a computer system. According to this paper, five components are necessary to state such rules:

users are the people[1] who want to use resources within the computer system. Users are needed so that a policy can state what a user is allowed to do.

resources are programs, services and data accessed by users. The policy specifies what resources a user can access.

operations are the specific operations which can be performed on a resource. Each resource type has a set of operations it supports. A

[1] or the processes on their behalf

policy must specify which operations the user is permitted to invoke on the resource.

authority is the legitimate power to make policy decisions. The authority can grant access to resources which are in its authority domain.

domain is a boundary containing resources or users.

2.2.2 Authorisation and Obligation Policies

The concept of policies was further developed and in [MoSl 91] policies were not only understood as rules for access control. In that paper, policies are intended to influence actions. Operations or actions are always characterised by a motivation and need authorisation to be performed.

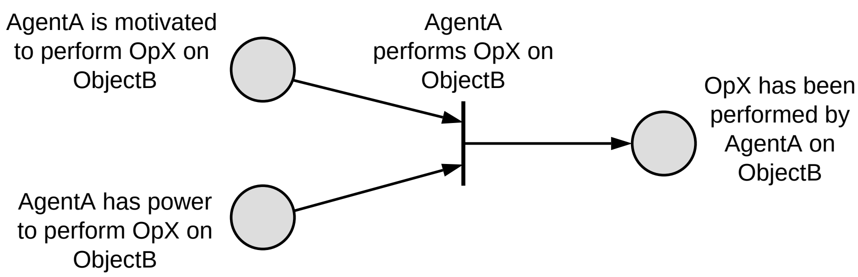

Figure 2.1: Precondition for an action [Slom 94a]

Figure 2.1 shows the precondition saying that both motivation and authorisation are necessary before an operation can take place. Although they share the same precondition, motivation and authorisation are seen independently of each other. Other facts shown in figure 2.1 are that motivation and authorisation must have the same subject, (i. e., AgentA) which performs the operation. Also, it must have the same target (i. e., ObjectB), which is affected by the operation. As a requirement, motivation and authorisation must be stated for the same operation, i. e. OpX.

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

Table 2.1: Structure of a policy [SlLu 98]

The general format of a policy is displayed in figure 2.1. Optional attributes are within brackets. The braces and the semicolon are semantic separators. Some attributes such as trigger, subject, action, target, and constraint may be comments, in which case the policy is considered a high-level policy and it is not possible to interpret it directly. It is made up by the following attributes:

identifier is the name of the policy.

mode To support the different characteristics of motivation and authorisation, a mandatory modality attribute was introduced. In recent publications the term *obligation* is used instead of motivation. Four policy modalities are possible distinguishing between positive and negative authorisation/obligation policies.

> **Authorisation Policies** specify which actions a subject is permitted or prohibited to perform on a set of target objects, or which pieces of information that are being monitored can be received. Therefore, there are positive (A+) and negative (A−) forms of authorisation policies. The positive form states that an activity is allowed and the negative one states that an activity is prohibited. To perform an action, explicit authorisation is necessary, which means that non-authorised invocations are forbidden by default.
>
> **Obligation Policies** specify, given a set of target objects, what activities a subject, i. e. a person or an agent, must perform in case of the positive form (O+), or must not in case of the negative form (O−). trigger events start the enforcement of positive obligation policies.

subject This attribute defines to whom the policy applies, i. e. which objects[2] are authorised or obligated (responsible) to carry out the policy goal within the limits defined by the policy constraints. Subjects are usually expressed as a domain of objects and a policy applies to all objects in that domain. Authorisation policies can have a domain expression of any (see figure 2.2) as subject.

action or policy goal, which is modelled by operations, specifies what must be performed in case of obligation policies and what is permitted in case of authorisation policies. Multiple actions can also be specified.

target specifies the objects on which the policy actions are to be performed. They are expressed with the help of domain scope expressions of objects. A policy applies to all objects specified by the domain scope expression.

constraint limits the applicability of a policy, e. g. to a particular time period, or makes it valid after a particular date. It must be evaluated and fulfilled before a policy is to have any effect.

exception attribute specifies alternative actions and is only allowed in the case of a positive obligation policy. With a failure of a positive obligation policy, the alternative actions are executed.

[2]persons and users are both treated as objects

parent and child are added automatically as part of the refinement process to create a policy hierarchy.

xref is a manually included cross reference list to additionally reference other policies, e. g. authorisation policies that grant permission for an obligation policy.

Syntax	Description
any	refers to all objects
$+$	set union
$-$	set difference
$\wedge$	set intersection
$*$, $*n$	if applied on a domain object returns a set that contains all direct and indirect members of the domain and the domain object itself; otherwise returns a set that contains the object itself – if an integer constant n is present, then the domain structure is traversed n levels down (thus limiting the search space)
@, @n	if applied on a domain object returns a set that contains all direct and indirect members of the domain excluding any domain objects; otherwise returns the empty set – if an integer constant n is present, then the domain structure is traversed n levels down
{ }	returns a set that contains the object to which it is applied (note that this is implicit for objects without braces as a shorthand)

Table 2.2: Domain scope expressions [Marr 97]

`A+ @/personnel/nurses {administer(analgesics)} x:@/patients/lung-diseases when (x.temperature>37) && (x.temperature<38.5);`
`A- @/personnel/nurses { validate } @/patient/discharges;`
`O+ on drugs_administered @/personnel { update } /drugs_db;`
`O- n:@/personnel/nurses {administer(_)} @/patients when n.state!=on_duty;`

Table 2.3: Authorisation and obligation policy examples [SlLu 98]

Table 2.2 lists the domain scope expression operators and table 2.3 shows some policy examples. Policies are modelled as system objects and there is a minimal set of operations defined, i. e. create, destroy, and query.

Negative authorisation policies and negative obligation policies are not semantically equivalent. Obligation policies are always interpreted by subjects, whereas authorisation policies are used by the target's support system to specify access rights.

2.2.3 Current Research Activities

The current research activities [LMSY 95, LuSl 97d, LuSl 97b, LuSl 97c, LSY 97] combine policies with a role concept of classic Role Theory. Affiliation to a role is represented by being member of a role domain. Policies apply to this domain. Policies are not entirely sufficient to model all requirements. Therefore, research is carried out on interaction protocols (e. g. delegation of tasks) and concurrency constraints (e. g. coordinating access to shared objects). Very recently, the domain model has been suspected of not being sufficient enough for modelling complex role attributes and relationships. The use of role classes with inheritance is investigated at the moment.

2.2.4 Discussion

The domain scope expressions used by the policies do not distinguish between an object and a domain object in all cases. Unwanted semantic changes may be the result when replacing a single object with a domain as the following example shows: /linux_user is a single object and used as subject in a positive obligation policy. When the policy is triggered by an event, the action executed has "exactly once" semantics. In case /linux_user is replaced by a domain object specifying a number of Linux users then the actions of the policy will be executed several times. No information on whether this ambiguity has been considered could be found. It may be resolved using a special operator or redefining an existing one, for instance by a @0 or $*$0 expression.

Although the policy approach described above is very powerful, it still suffers from certain restrictions. Constraints other than access rights are not entirely considered (e. g., "the last three passwords must not be reused"). In many cases, there is probably a way to describe other constraints on the basis of permissions, but the meaning of such (a number of) policies is not easy to understand.

Only in the case of positive obligation policies can additional actions for an enforcement failure be specified. In all the other cases it is not possible. Especially management operations must handle failures carefully. The time when a policy fails cannot be specified. The notation and semantics should be enhanced to allow this. In the remaining chapters of this thesis we will see that this expressiveness is important in order to support metapolicies.

Some aspects of the notation are not straightforward to guess. For instance, passive obligation policies are triggered by events, negative ones are not; negative obligation policies are only considered by subjects, negative authorisation policies are considered by the target's support system, and positive authorisation policies are used by both. All this may be a consequence of the initial research interest which focused on access control.

2.3 Policy Work by Koch

The policy work by Koch follows in many aspects the approach of the Imperial College. Additionally, in [Krel 95, Koch 97, KKK 96] policies are structured in a hierarchy of three levels: refinement level, goal-oriented level, and operational level. The description at each level varies resulting from the different level of detail in the policy development.

requirement level At this level, policies are described in prose. They describe the objective without considering technical constraints. However, the specification is guided by finding policy related information such as entity-relation, required authorisation, system invariants, and domain and object definitions.

If it is clear that an objective cannot be met, this fact should be expressed. For further development, a best-effort approach is used. The refinement process is finished within one level whenever the policy provides sufficient detail, allowing a simple transition to the next level.

goal-oriented level The policy details are increased at this level. The abstract requirements are transformed more closely to the actual management model and the entities are bound to the existing environment. A template is used to achieve this step. The template consists of the core attributes of a policy described in subsection 2.2.2. The refinement work can be summarised as follows: desired system behaviour describing invariants gets translated into management activities; triggering events and the subject are named; constraints are developed, which must be fulfilled before an action is initiated; the modality (obligation[3]/authorisation) of the policy is chosen.

operational level The policy representation at this level is used for enforcing the management activities. A policy at the goal-orient level is split into two parts, the actual policy and necessary event definitions. Policies are described by a *Policy Definition Language (PDL)*. Events used by policies are defined by an *Event Definition Language (EDL)*.

The PDL combines the parts necessary for a policy definition. These are the event definition, the domains, the management interfaces, and the desired management activity. In addition to the policies as defined by the Imperial College, the PDL supports the explicit generation of events in successful and

[3]In contrast to obligation policies by the Imperial College, negative obligation policies are not used.

unsuccessful cases. The intention of this feature is the realisation of policy *chains*. New events can be triggered by a policy depending on the result of an action.

With EDL, the events are defined. The EDL influences the behaviour of the monitoring components and allows the creation of timing events. An event composition is also possible allowing basic event correlation. The chaining mechanism can be used in combination with the conditional creation of composite events to design complex activity chains.

Discussion

Koch follows exactly the paradigm of authorisation policies of the Imperial College. The restrictions in this area, as described before, are consequently present here. Again, success or failure of a policy enforcement is not specified in the policy. Adding a postcondition as a counterpart to the precondition in the constraint statement would be a solution. Otherwise, the policy is dependent on the results of an action it invokes. This is a restriction when using policies in heterogeneous environments. Applying them in homogeneous ones, as Koch is doing, will not cause these kinds of problems. Negative obligation policies are not used. The policy notation is therefore more consistent as the one described in the previous subsection.

2.4 Constraint-based Policies by Goh

Recently, Cheh Goh presented a generic policy approach [Goh 97], which is slightly different from the ones presented before.

Central to this policy concept are the states of a managed system, object, or resource and the formulated constraints on these states or state transitions. With the help of constraints, the desired behaviour of the states of the managed objects is described. A system is compliant with the policies if it satisfies all of the constraints on the system and state transitions.

Figure 2.2 shows permissible and impermissible states and state transitions according to stated policies. There is a distinction between passive and active policy enforcement, which is described later in more detail in subsection 2.4.2 and 2.4.3.

In comparison to the approaches influenced by the policy work of the Imperial College, in [Goh 97] no categorisation in authorisation and delegation policies is made. Instead, his view treats policies as constraint descriptions and these are powerful enough to include the approaches presented in the sections above. We will discuss this in section 2.4.2 and 2.4.3 in more detail.

In general, actions are not implicitly considered as part of a constraint-based policy as they are, for example, in Sloman's delegation policies. The reason for this is that, given the constraints, there are a potentially large number of valid ways to move a system into a state where it is compliant with

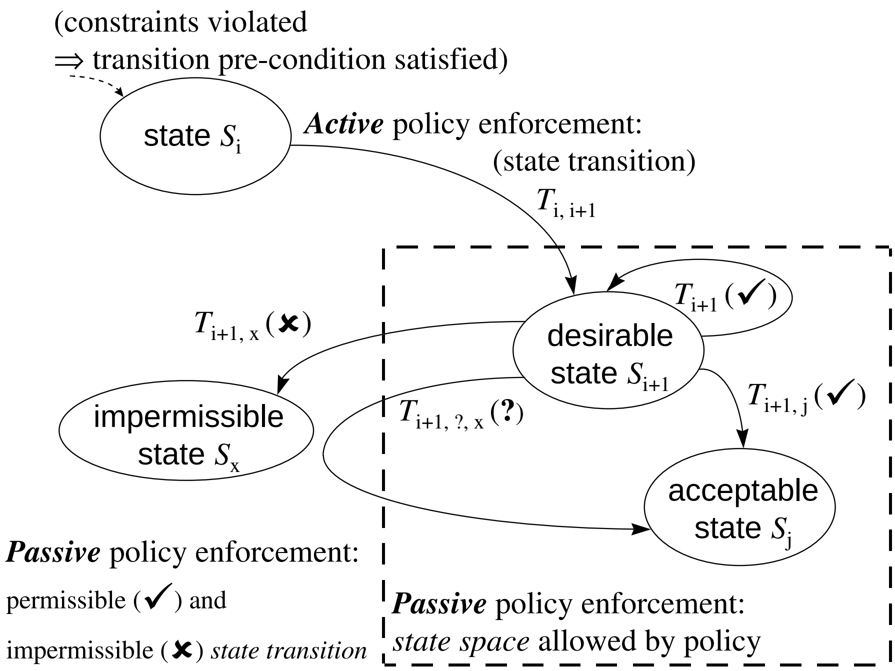

Figure 2.2: Active and passive policy enforcement in a system [Goh 97]

the set of stated policies. A (passive) policy does not restrict the enforcement to fulfil the constraints through a specified algorithm which would be encoded in the policy itself. So called *implementables* are added to active policies during the final phase of the refinement process. In the following subsection, this is described in more detail.

2.4.1 Refinement

In this subsection, experience and additional thoughts regarding the refinement process are presented.

Using constraints to describe policies supports the idea of policies for management, where policies are used to achieve an objective. Therefore, it is important to specify the desired objective in the beginning. Constraint based-policies are used for this purpose. As in the other approaches, it is necessary to refine (high-level) policies, because using constraint-based policies does not change the fact that an objective is vague at first.

The objects used in the constraint statements must be well known and provided by an appropriate information model of the infrastructure. In discussions[4] and work on the refinement process of constraint-based policies it became clear that the possibility of adding new objects and relationships

[4]Thanks to Cheh Goh, Adrian Baldwin and Marco Cassasa Mont from the Hewlett-Packard Laboratories, Bristol

between objects to the information model during the refinement of a policy would make this task easier. At least at the moment, an applicable generic information model has not been developed for most management needs. This possibility of adding objects would not be as important, if in the future, customised or branch-specific information models (as for insurances, banks, production plants, warehouses, etc.) were available. These information models could be the basis for applying predefined branch specific policies.

The strong orientation of the policy approaches towards obligating actions as described above results in a refinement process where mostly actions are refined. In the constraint-based approach, the focus is more on the constraints. This includes conditions on states and state transitions and thus on objects and relations between objects. Typical tasks in the refinement of a constraint-based policy are the refinement of objects. This could be supported by the information model with object hierarchies or inheritance combined with specialisation. Otherwise, additional constraints are used to specify a subset of the initial set of objects. This could be accomplished for example with domain scope expressions or constraints on attributes of the objects. Refinement of relationships between the objects must also be done. When the refinements are treated like objects (entities) hierarchical structuring and inheritance with specialisation could again be used for accomplishing this task. In addition to refining the relationships and objects, the constraints themselves are to be refined. When refinement of already specified objects and relationships is not enough, needed objects and relationships are added. Splitting of a policy into a number of policies for the purpose of refinement is also possible. At the end of the refinement process, a policy is a passive policy or an active policy with operations.

The operations as the final implementation of an active policy are added during the final phase of the policy refinement process. The operations added to an active policy are called *implementables*. An operation is sufficient if it fulfils all of the constraints of the policy. Choosing the operation near the end of the process minimises unwanted side effects as a result of choosing inappropriate operations, because at this level of refinement most of the constraints are known.

Support from an interactive planning system could provide suggestions as to which set of operations is able to transform a system from one state into another. Such a planning system is given constraints, and by satisfying these, a set of possible actions is calculated. Using such a system would help to prune the search space of actions in the refinement of policies [BLMM 97, BLMM 98, BaBa 98, BaMi 98].

2.4.2 Passive Policies

Passive policies do not include implementables. They have the character of an information repository for the enforcement part of the policy management system or planning system mentioned in the section above.

Passive policies contain information about which states are desirable and acceptable, as can be seen in figure 2.2. Passive policies define an allowed *state space*, displayed by the square enclosed by the broken line. The diagram shows a system in the state S_{i+1} from which the transitions T_{i+1} and $T_{i+1,j}$ are allowed. These transitions are allowed because they will not leave the permissible state space during their execution and they bring the system into a state which is also in the allowed state space. The transition $T_{i+1,?,x}$ leaves the permissible state space during the transition before returning to S_j. Whether transition $T_{i+1,?,x}$ may be possible or not, depends on the actual (meta)policy. Therefore, it is marked with a question mark. Potential problems and questions may arise:

- If the transition fails during the execution, the system may not be compliant with the policies because it could have left the allowed state space.

- *How far* is an action allowed to leave the permissible state space and when must the action after the transition bring the system back to an allowed state?

- Whether an algorithm halts is not decidable, so it could run forever, and thus the action could stay outside the state space.

- It is harder to track actions which are allowed to leave the allowed space. Therefore side effects are more likely, and an altered operation would not be recognised so easily.

A possible solution for some of the questions and problems could be the realisation of state transitions similar to transactions used in databases where it is possible to roll back a change to an inconsistent state in case of a failure. This is commonly known as the *two-phase commit protocol*.

As mentioned in the beginning of this section, passive policies do not have actions, but they are used to allow or deny actions. This can happen in various ways: Either the information is translated into a configuration of a resource, for instance an access control list, or the policy system acts like a wrapper or proxy for the desired activity. It grants the activity if it is compliant with the specified passive policies. The activities must be well known when using this mechanism.

The first case allows for very quick operation. The latter will be much slower but has a greater flexibility, because complex constraints, which cannot be translated into static configuration files, are possible. Another advantage is the ability of recognising runtime conditions which are, in our understanding, only a particular instance of states. A mixture of these two extreme cases is also possible. In section 6.2.3, this is discussed in greater detail for the realisation of passive (meta)policies in an existing management system.

2.4.3 Active Policies

Active policies are called *active* because they make use of implementables in the policy description, either implicitly or explicitly. The implementables serve as the active actions and make up the transition plan being executed during enforcement. This transition plan is being developed to change a defined set of states. In addition to the transition plan, active policies include constraints which define the preconditions and postconditions of the policy. In section 2.4.1, the general refinement of an active policy is discussed.

Active policies, as opposed to passive ones, can react to perturbations of the permissible state space. The perturbations can have various reasons: a failure of a hardware or software component, or the installation and configuration of new or existing hardware or software. Active policies are used to make a system compliant with the entire stated policies again.

In the policy precondition it is specified what triggers the enforcement of the policy. After the enforcement of the active policy, the system must satisfy the postconditions, otherwise the enforcement of the policy has failed. The postcondition has the character of a passive policy, because the contained constraints must specify a subset of the permissible state space. If this was not the case, the active policy could cause a transition to an impermissible state. Such a policy would actively transfer the system to an unwanted state, which is against the idea of its intended management purpose.

2.4.4 Discussion

In comparison the authorisation policies introduced by the Imperial College, the passive policies are a superset of them, because an authorisation policy itself is a policy which specifies a constraint, the access right over the state of an object, i. e. the access of a subject to a target. This is only a special case of a passive policy where the semantics of the constraint is not limited to access rights. Passive policies may not have a subject, because they mainly describe a state space[5].

Active policies with implementables can be considered as a different view upon obligation policies. The active policies are more focused on state transitions than the other policy approaches. Consequently, they explicitly consider postconditions as a description of the desired state after the execution of the implementables. This makes it possible to verify whether an enforcement was successful. Splitting of state transitions (i. e., adding of an intermediate state) is possible and is part of the policy refinement.

[5]Authorisation policies express this by having the domain scope expression “any” as subject.

The description of the desired state and the actions to get there could be seen as a more complete description of policies, because it expresses more clearly the goal of a policy. In the case of obligation policies this information can (hopefully) be restored by the policy hierarchy.

2.5 Summary and Terminology

This chapter introduces the concept of information system management policy and presents a survey of common policy approaches. Differences and similarities between these approaches are discussed. Comments on specific problems or ambiguities are given.

All policy approaches have in common that they explicitly consider refinement as an important process. The quality of this process differs with the policy approach. In section 4 other processes are described. In the next chapter an overview of metapolicies is given.

Terminology In the remaining chapters of this thesis, the terms *active policy* and *passive policy* are used because the approach by Goh includes the expressiveness of the approaches by the Imperial College and Koch. The terms for the attributes a policy includes are used as introduced in subsection 2.2.

Chapter 3

Previous Work on Metapolicies

This chapter presents an overview of the understanding of metapolicies in the literature. Each section describes a metapolicy concept introduced by a different author or institution. As will be seen, each described metapolicy concept has a special area of interest. For this reason, they are described separately without introducing a common understanding. The chapter ends with a summary and a discussion of the presented concepts.

3.1 Metapolicies by Hosmer

Two publications, [Hosm 91] and [Hosm 92], by Hilary H. Hosmer are the only publications devoted entirely to the concept of metapolicies. All other publications mention metapolicies mostly as a marginal topic.

By Hosmer, policies are seen in the context of large, complex, and interrelated trusted systems and understood as "a set of constraints established by an accepted authority to facilitate group activity." Special areas of interest are security policies, which are "the plans of an organization to meet its security goals." Subpolicies are policies which govern concerns of a broader policy [Hosm 92].

The metapolicy concept by Hosmer is motivated by the desire to turn implicit information in policies into explicit information and to incorporate a set of rules for guiding the application of multiple policies. In [Hosm 92], they are defined as follows:

> *Metapolicies are policies about policies. They make the rules and assumptions about policies explicit rather than implicit and coordinate the interaction of multiple policies.*

Two aspects are mentioned, the implicit information of policies and the coordination of multiple policies. Both are presented in the following subsections.

Policy Description	Data Type	Length	Criticality	Req. Signer	Modifier
Policy Name	Alphanumeric	20	30	Scty DOD	SSO
Policy Type	Alphanumeric	5	30	None	SSO
Authority	Alphanumeric	30	50	Scty DOD	SSO
Start Date	Date	6	20	President	SSO
Expiration Date	Date	6	25	President	SSO
Informal Model	Alphanumeric	900	20	None	SSO
Formal Model	ZED	1500	40	Sys Manager	SSO
...					

Table 3.1: Policy Description Metapolicy [Hosm 92]

3.1.1 Explicit Information

Implicit information exists in most cases either as an informal or as a formal statement. Examples of implicit information are the scope, description, structure, interrelationships, control, and rendering of policies. Metapolicies make this information explicit. The following kinds of metapolicies which specify implicit information are mentioned by Hosmer:

Policy Description Metapolicy defines the structure of the policy presentation, the elements which make up the policy, the name, type, and length information of the elements, etc. Table 3.1 shows an example of a Policy Description Metapolicy. Besides other possible information, information about the required signers and modifiers is contained in the example. This indicates who may approve changes to a policy element and who may modify the element. In the last line of the example metapolicy these are, for instance, the System Manager and the System Security Officer (SSO). The criticality code is an indicator for the impact a change will have on the rest of the policy and/or the security of the system.

Policy Constraint Metapolicy The constraints put on a policy are specified with a Policy Constraint Metapolicy. It may consist of the time of execution, the restrictions on the application domain, time limitations like expiration date, exemption of certain users or roles from the policy, and may specify whether the policy must be executed in combination with other policies.

Organization Control Metapolicy With metapolicies of this kind, information needed for working with a policy in an organisation is made explicit . In table 3.2, an example of an organizational control metapolicy is shown. With the help of this metapolicy, for example the owner and the creator of the policy is described, when its expiration date is, and what processes are used for its distribution, renewal, and modification. Also included may be the policy assurance status, legal status, and the source of the policy.

Automated Information System (AIS) Metapolicy With this kind of metapolicy, information necessary for the computer-oriented processing of the policy is specified. It contains details needed to describe and control the implementation of the policy and it may include constraints on the implementation mechanisms, requirements for configuration management and audit, etc.

Domain Interface Metapolicy This metapolicy specifies rules when data labelled for one policy domain must be transferred to another policy domain. They are used for the automated transfer.

3.1.2 Coordination of Policies

In a system with multiple policies, there are relationships between these policies. Therefore, metapolicies are specified which control concerns like conflict resolution, execution precedence, etc. They may also make implicit information explicit, but, in addition to the metapolicies described before, they focus on more than one policy. This is how Hosmer itemises them into separate metapolicies:

Policy Relationship Metapolicy A relationship between policies is described by a metapolicy which specifies the policies and the concerns involved in the relationship. Table 3.3 shows an example of a Policy Relationship Metapolicy. As can be seen from the example, it is specified whether the relationship is hierarchical or collegial. It describes the sequence of execution, the precedence in case of a conflict, the importance of the relationship to the security of the system, the creator of the relationship, and the person authorised to change it. Many other items could be included.

Subpolicy Interaction Metapolicy Policies operate in concert with many subpolicies. The relationships to the subpolicies are made explicit with the Subpolicy Interaction Metapolicy. An example of such relationships is depicted in figure 3.1. As can be seen in these figure, the Access_Control policy is interacting with policies named MAC and DAC[1]. Since this metapolicy describes subpolicy interactions between separate policies, these could also be specified with the Policy Relationship Metapolicy.

Site-Specific Metapolicy The Site-Specific Metapolicies are used to specify policies which control administrative or domain-wide policies for a particular user site.

[1] MAC stands for Mandatory Access Control, DAC stands for Discretionary Access Control, and I & A stands for Identification and Authentication

POLICY NAME:	Organizational Control
POLICY TYPE:	Metapolicy
AUTHORITY:	AIS Policy Center
CHANGE PROCESS:	Two SSOs with written approval from AIS Policy Center
UNDERLYING POLICY	
Policy Name:	No Read Up
Source of policy:	Executive Order 123456
Legal Status of policy:	Mandated by federal law
POLICY PEDIGREE	
Owners:	DoD
Creator:	Defense AIS Security Policy Center
Date Created:	1962
Expiration Date:	1992
Authors:	John Smith, Sarah Jones
Reviewers:	MITRE, Aerospace
ASSURANCE	
Policy Criticality:	High
Assurance Level of Policy:	B3
Policy Evaluator:	Commercial Evaluation Center #3
APPROVAL PROCESS	
Final Authority:	President of the USA
Approving Organizations:	US Department of Defense, Army, Navy, Marines, Air Force, Coast Guard, Defense Intelligence Agency, DARPA, Joint Chiefs of Staff
Approval Sequence:	Approving organizations give their approval in parallel, then it goes to the President
POLICY IMPLEMENTATION	
Effective Date:	1963
Application Scope:	Applies to all USA classified data
Oversight Committee:	Joint Chiefs of Staff Committee 234
RENEWAL	
Renewal authorization:	President of the USA
Renewal terms:	3 to 10 years
Renewal Process:	Service and JCS approval
MODIFICATION	
Authorization for modification:	President of the USA
Policy modifier:	US AIS Security Policy Center
Process for modifications:	Review by all services
Last Date Modified:	1985
DISTRIBUTION: (Unlimited/Limited/Controlled)	
Unlimited	
PUBLICATION DATA:	
Publisher:	DOD Publications Center
Document:	Military AIS Security Policy
POLICY USED IN:	
Government Documents:	DOD 654321, AF 802-456
Commercial Hardware:	all MLS products
Commercial Software:	all MLS products

Table 3.2: Organizational Control Metapolicy [Hosm 92]

Policy Relationship Metapolicy	Policy 1	Policy 2
Policy Names	MAC	DAC
Relationship (Parent/Child/Colleague)	Collg	Collg
Execute (With/Before/After/Not)	Before	After
Precedence Level in this relationship	100	50
Criticality of relationship	80	
Creator of relationship	X. Jones	
Authorized Modifiers of relationship	Sec. DOD & SSO	
...		

Table 3.3: Policy Relationship Description Metapolicy [Hosm 92]

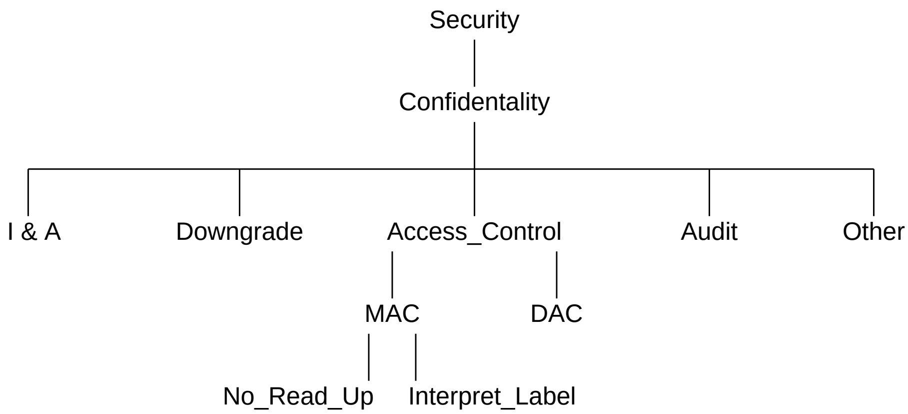

Figure 3.1: Hierarchical Subpolicy Structure [Hosm 92]

Multipolicy Coordination Metapolicy This metapolicy coordinates the multiple security policies which are present in a system at the same time. The user's priorities and tradeoffs will be considered. These may be a complex metapolicy with many levels, domains and implementation forms.

3.1.3 Properties of Metapolicies

Metapolicies by Hosmer are primarily introduced to provide control for the organisation, the automated information system, and the security subsystem.

They differ vastly in scope and significance. There are several metapolicies for every (security) policy and one metapolicy may coordinate many policies. There are general rules (guidances) which always apply to the creation of metapolicies. Additionally, certain situations may have certain rules, but others may not. Making implicit information explicit happens on a subjective basis and agreement on this must be found.

As a result of the definition and the aim of introducing metapolicies, there will be metapolicies about metapolicies.

Metapolicies can consist of a single value or can be implemented in modular and layered data structures. These layers may correspond to the layers of the organisation, the layers of the computer system, or layers of security policies.

Changes or additions of metapolicies are security-relevant events, because they are as critical as security policies. Therefore, metapolicies must be audited as in the case of security policies.

Metapolicies are considered to be very important in trusted systems, because they control key functions like the processing of policies and the coordination of policies. They support the control of policies, help to standardise the policy formats and improve the interchangeability of policies.

The development of graphic techniques, such as those used in the field of data bases, can help to visualise the metapolicy relationships, groupings, interactions between metapolicies and between policies and metapolicies

3.2 Metapolicies by Kühnhauser

As metapolicies by Hosmer, Winfried E. Kühnhauser from the German National Research Center for Information Technology (GMD) published three papers concerning metapolicies in the context of security policies [KüvKO 95, Kühn 95a, Kühn 95b].

Kühnhauser introduces metapolicies as a consequence of the demand for cooperation between policies from different domains. Large and complex computer networks connecting several independent organisations have security domains and each domain has security policies. He follows Hosmer's arguments for metapolicies focusing on the coordination aspects. He argues that cooperation among a complex network requires the definition of the relationships between the security policies involved. This concept is referred to as metapolicies and [Kühn 95b] defines them as follows:

> *Metapolicies are policies about policies. They create a framework for the complex coexistence of several security policies containing rules for interfacing, cooperation and conflict resolution.*

It is not clear which constraints for interfacing and cooperation between policies are needed in the end, because the way of communication between policies is not well understood at the moment.

With regard to conflicts and coordination between policies, metapolicies are used as rules interfering with the policies of different domains. Metapolicies act on policies, therefore a public interface is required to allow metapolicies to intervene. Clearly they must be agreed upon in advance by both parties.

	i_1	i_2	i_3	$\cdots$	i_n
i_1		crf	crf	$\cdots$	crf
i_2			crf	$\cdots$	crf
i_3					crf
$\vdots$		not used			$\vdots$
i_n					

crf: A Conflict Resolution Function or Relation

Table 3.4: The Conflict Matrix [KüvKO 95]

For controlling conflict and cooperation of policies the concept of a *Conflict Matrix* and a *Cooperation Matrix* is introduced [KüvKO 95].

The elements of the Conflict Matrix are *conflict resolution functions*. This allows the definition of a function for a pair of policies defining a resolution strategy. An example for a conflict resolution function is a partial order on the two policies. In the case of conflicting access control policies, simple logical `and` or `or` functions can be used. In principle, other functions are possible.

The Cooperation Matrix is very similar to the Conflict Matrix. The only difference is, that the elements are describing the *causal precedence functions*. With their help, a decision can be made whether a policy must be enforced before another.

It is characteristic for metapolicies is that they may be treated like any other ordinary security policy, apart from the fact that the objects within their domain are security policies instead of application system objects.

3.3 Metapolicies at the Imperial College

In [LuSl 97d], [LuSl 97a], and [LuSl 97b] metapolicies are used as a way to specify logical predicates over different policies, which were described in section 2.2.

They are introduced because conflicts may arise between policies applied to the same object processing different management functionalities, or policies are applied to objects which are members of several domains.

The metapolicies are *policies about permitted policies* and their only purpose is to specify application specific consistency constraints pertaining the contents of policies in one domain. They can be expressed as logical predicates applying to sets of policy objects within a domain. Table 3.5 shows the predicate specifying a conflict as an example where a conflict of duties exists: *The same manager cannot both authorise payment and sign the payment cheque.*

∀P1, P2 ∈ <domain_scope_expr>

fail ⟵ intersect(P1.subjects, P2.subjects) ∧
belongs('payment', intersect(P1.targets, P2.targets)) ∧
belongs('authorise', P1.actions) ∧ belongs('sign', P2.actions)

Table 3.5: Predicate of a metapolicy [LuSl 97b]

At the Imperial College the predicates are translated into Prolog statements in order to find conflicts in a developed policy-based system.

3.4 Summary and Discussion

All three presented metapolicy concepts stick more or less exactly to the meaning of *meta*, expressing that metapolicies are policies about policies. Each concept was introduced for solving a special problem or supporting a specific area of interest. This is the result of using policies in very different contexts and for various purposes. As a result of this, each concept is very customised according to the context of the policies investigated.

Although the specification *policies about policies* implies a huge area of valid metapolicies, some metapolicies may be considered as policy templates. Especially the Policy Description Metapolicy by Hosmer includes much information which can be seen as part of a policy template as described in [Wies 95]. Other information like the Requested Signer may depend on every particular policy. Therefore, specifying this information in a Policy Description Metapolicy for all policies may be not such a good solution. This will ensue many exceptions.

Metapolicies by Hosmer only respect security concerns with organisational aspects necessary to achieve a certain level of security. This includes metapolicies for the automatic processing of security policies, but other aspects have not been investigated.

Making implicit information explicit also helps to clarify things, but Hosmer creates ambiguities with Subpolicy Interaction Metapolicy and Relationship Metapolicy. It is not clear which relationship concerns are described in the one or the other metapolicy type. Furthermore, describing the same aspects redundantly in several independent metapolicies does not help to clarify things. Another example is the expiration date which is specified in three different kinds of metapolicies.

Kühnhauser introduces the Conflict Matrix to be able to take a decision in policy conflicts of policies of different domains. This mechanism does not seem to be adequate for handling conflicts in such a complex scenario. The context during the enforcement of the policies is not considered. This has also to be taken into account as the following scenario makes clear: A policy having a special domain as subject will be enforced in case of two conflicting policies. It is also obvious that the case where several conflicting policies have the same subject (i. e., more policies are in conflict) must be handled differently. A more flexible approach is therefore necessary.

Using analogous arguments it can be shown that the Cooperation Matrix is also not sufficient.

In contrast, metapolicies at the Imperial College are used to specify the conflicts. They are additional constraints applied during the execution of an application. For this reason, the targets are not necessarily policies.

All metapolicy concepts have in common that they are seen in the context of conflicts. It also seems to be clear that fact that using policies for various scenarios and reasons implies additional information and control.

In chapter 5, the concept of metapolicies is investigated in more detail, and their context of application is enlarged.

Chapter 4

Policy Life Cycle

In this chapter, the basic processes involved in the policy creation up to its actual enforcement are presented. The precise understanding of the phases and processes involved is crucial for the understanding of policy management, which leads to the concept of metapolicies developed in this thesis.

I extracted the functionality required for the application of policies and identified the functional areas. Based on this analysis I developed an enhanced policy life cycle presented here.

Before going into the details, however, I will first give a brief summary of the context in which policy life cycles and related processes were mentioned. The chapter closes with a summary and discussion of the presented material.

4.1 Previous Work

Looking at recent publications in the area of policies, present research activities appear to focus more on the architecture of policy systems [Marr 97, Koch 97], or deal with the application of policies in different areas of management [LuSl 97d, SlLu 98, Heil 98]. Hereby, mainly (technical) enforcement concerns are investigated. As a consequence the policy life cycle, and the processes involved in the life cycle have often been neglected in the architecture of a policy system. Whenever they were mentioned, they were used to model the application of policies.

[MSY 95], [Marr 97] and [Wies 95] use a similar but limited, policy life cycle. [MSY 95], [Marr 97] use it mainly to describe some constraints for editing, distribution and deletion of policies. In brief, a policy can only be edited and deleted when it is not enabled and is not in the host system; in all other cases editing is done on an automatically created copy.

[Wies 95] additionally derives from his life cycle that policies must be able to send notifications when the target or the domain has changed, which lead to a change of the policy configuration and its monitoring part.

[Goh 98] examines a Policy Evolution Life-cycle focusing on the development (and especially the refinement) and the stages a policy has to pass from the initial organisational requirement establishment to the mapping for configuration and the verification of the policy enforcement. Through this development process and the different people involved in this process having various demands, a need for policy management arises.

4.2 Policy Supporting Processes

The use of policies for the purpose of (automated) management entails several steps in which the policy is created, modified, reviewed, tested, processed, and finally enforced. These steps are accomplished by supporting processes, which will be described in this subsection.

First, a general description of characteristics of these processes is given, whereby the differences between active policies and the supporting processes are clarified. This differentiation will be needed in the later chapters. After that, a functional overview is given, and each of the supporting processes is described.

4.2.1 Supporting Processes and Active Policies

In this thesis, the term *process* is equivalent in meaning as a functional unit executing something in a certain area. In the context of workflow management systems, it is called *task*. This functional unit may or may not be realised in the form of an executable computer program or algorithm. In most cases it is a mixture of both, a computer program combined with human interaction, as, for instance, in the case of complex decision making.

Each process will be looked at as a set of actions belonging to a functional area. These functional areas are extracted by the investigation of the policy approaches described in chapter 2 and examined additional organisational and practical concerns.

Table 4.1 shows a comparison of characteristics of supporting processes and active policies. These will be discussed item by item beginning with the first one.

- There are several processes which are necessary in a policy-based management system. Often they tend to be rather complex such as the refinement process. In contrast, active policies, being the result of such a refinement process, contain only a comparatively small number of activities necessary for the enforcement of a small management subgoal.

- As a result of this, the supporting processes more often require human interaction, whereby the active policies may only need human intervention in case of an enforcement problem. Some enforcement problems

supporting processes	**active policies**
• complex tasks performing a certain area	• only management activities
• often require human interaction	• may require human interaction in case of an enforcement problem
• description in various formats	• single representation format
• use multiple sources of information	• use well known notification channels
• process various kinds of information	• triggered by notifications
• move policies	• cause the invocation of actions
• multiple (ambiguous) roles responsible	• single subject responsible for enforcement
• consider organisational aspects	• organisational aspects already considered
• complex execution structure	• strictly specified enforcement
• review of process and produced data	• postcondition to check its enforcement
• unknown side effects	• explicit consideration of side effects

Table 4.1: Comparison of supporting processes and active policies

can be considered in advance. So, in these cases no intervention is necessary. The ratio of human processing and self-sufficient program processing varies between the different supporting processes. For example, the enforcement of a policy can be done more automatically than the refinement process.

- Processes are described in various formats. The spectrum of format ranges from plain text – as a description for the person with a specific responsibility – to implemented algorithms where full automation is possible. In contrast, once fully refined, active policies will be specified in a policy description format.

- Another consequence of the complexity of the supporting processes is that they use multiple sources and various kinds of information. For instance, the distribution process makes use of information such as the domain structure, the policies to distribute, constraints on the policy distribution, etc. Active policies use only the notifications by which the policies get activated. These are received through well known channels.

- Another difference is that the supporting processes in their entirety make the application of policies happen. They and the policy supporting processes are necessary to create, modify, distribute, etc. policies. The active policies invoke actions for the purpose of management. Only the actions triggered by an active policy may use additional information and information sources as the actions can be part of a large process, for instance an agent.

- In the case of the supporting processes, a number of roles can be involved for accomplishing a task. For instance, the different roles are responsible for refining policies in different areas such as security, performance, etc. An active policy has strictly a single subject responsible for its enforcement. This is explicitly specified in the policy representation.

- The supporting processes must take organisational aspects into account, especially the delegation and refinement process. The delegation process as described in section 4.2.3 almost exclusively has to deal with organisational aspects. In the refinement process organisational principles and constraints (such as costs) are incorporated in the created policies. Another example is the test process which has to prove aspects for which the organisation needs logging data, e. g., required by an insurance company. In case of active policies, the organisational aspects are already considered since they are the result of the refinement process.

- Processes have a complex execution structure. For instance the refinement process can be interrupted when a test or enforcement problem occurs. In comparison to this, the active policies have a strictly specified enforcement task which must be carried out by the enforcement process. The execution structure of policies can become complex by using chaining, i. e., the triggered enforcement of a number of policies. Inter-policy communication is not considered in this thesis, because very little is known about the benefits that can be derived from the complexity.

- The success of the enforcement of the active policy is checked by postconditions which are part of the policy description. Usually, no direct equivalent exists on the side of supporting processes. The closest to that is the reviewing of a process and its produced data. Policy descriptions which can be syntactically checked are produced as a result of the refinement process.

- Active policies cause changes on their targets for the purpose of management and therefore, the consideration of side effects of these changes is crucial for the successful enforcement of a policy. In case of processes, the side effects are usually not investigated in detail. The test process may be an exception, because it can interfere with the enforcement of policies.

For all this reasons, active policies and processes are not considered as equivalent concepts. This will be used in subsection 5.3.

4.2.2 Functional Model

In figure 4.1, the functional model of the supporting processes used in the remainder of this thesis is depicted in OMT[1] style notation [RBP+ 91]. It

[1] Object Modelling Technique

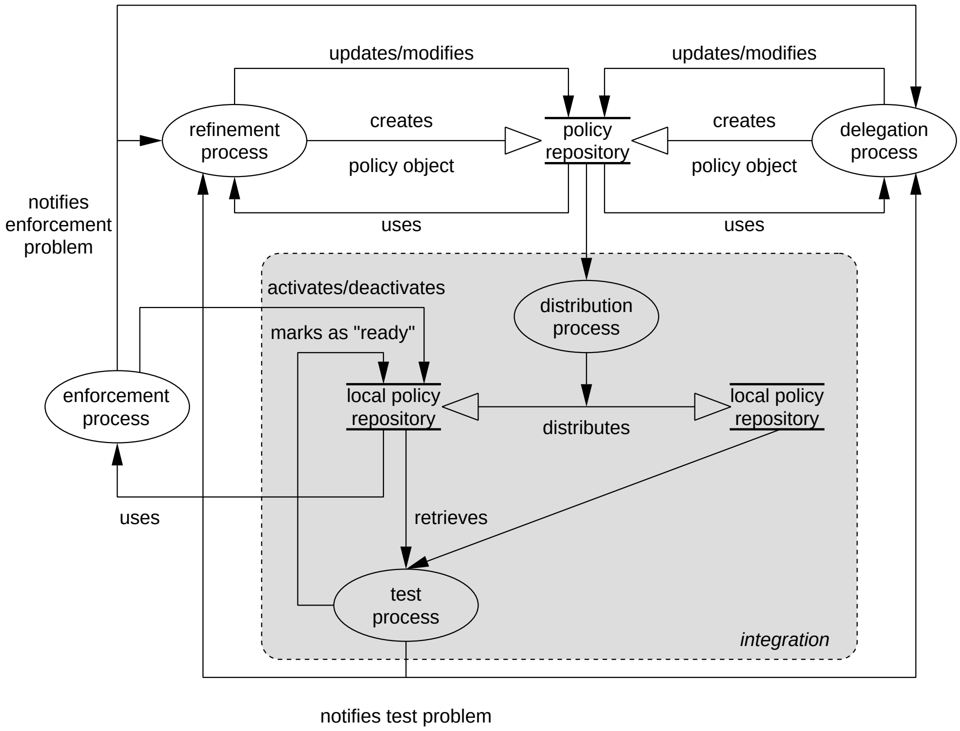

Figure 4.1: Functional model of the supporting processes

focuses on the processing of policy descriptions and policy objects. Other data objects additionally used by the processes, as mentioned in section 4.2.1, are not shown in the figure. It does not show any architecture, nor does it make any assumptions about it. The processes are looked upon as functional units, not as architectural components and may be implemented as separate components. This functional model provides the basis for the enhanced policy life cycle which will be introduced in section 4.3.

Figure 4.1 shows two types of policy data stores, a *policy repository* and a *local policy repository*. In the first one, all existing policies are stored. It serves as a global policy library and will be used by the *refinement process*, the *delegation process*, and *distribution process*. Not considered here, although necessary, is a mechanism to query and retrieve a policy by certain keys or characteristics, e. g., security policies, domain specific policies, etc.

The other type of policy stores contains policies in an enforceable representation and is used by the *enforcement process* and the *test process*. They are local, i. e., near the enforcement processes, because potentially the managed objects are widely distributed. Therefore, the enforcement processes and test processes are created close to the managed objects. Another reason for having them is that certain enforcement processes need a different policy representation. Such respresentation may be generated during the distribution process and stored at a local policy repository for these enforcement processes.

The grey area enclosed by the broken line depicts the processes and data stores involved in the *integration* of the policies.

Processes and constraints which would be involved in policy interchanges are not considered, because little is known about them. Exchanging policies over policy domain boundaries will add additional complexity as it will require a unification, or translation of the entities the policies use. This is necessary, because there will be no assurance that the entities exist, or that they are organised in the same structure as in the initial domain for which the policy was created. Another aspect adding complexity is the possibility of multi-domain conflicts, which are conflicts resulting of the integration of a policy from a "foreign" policy domain.

In the following subsections each supporting process is described in more detail.

4.2.3 Delegation Process

In the delegation process, various aspects (such as transfer of the refinement task, transfer or delegation[2] of responsibility and approval of the final policy) are included.

The need for management is almost always a consequence of a decision taken in the management level of an organisation. Policies can be used to specify a management objective in a high-level fashion. The refinement process tries to narrow the objective down to concrete low level policies achieving the initial principle. Failure due to ambiguity as a result of different reasons are not investigated here.

An example which is part of the delegation process follows: The customer relationship officer creates a high-level policy governing transmission of customer data, because he wants to offer this kind of quality of service for customer privacy reasons. He delegates the refinement of the high-level policy to the system security officer, who creates additional policy objects needed for the actual enforcement. After the system security officer has completed his refinement work, the created policies derived from the initial one go back to the customer relationship officer. He reviews them to ensure that they fulfil his demands.

The additional data of the delegation process should be stored within the policy repository for clarity, if not in the policy itself. The delegation process therefore updates and modifies policy descriptions, and uses them, for instance for approval purposes. This can be done in all steps of the refinement phase as different subjects may be responsible for various refinement

[2]Delegation and transfer of responsibility are used as follows:

$$\forall\ \mathsf{A\ delegate\ (resp\ R,\ B)} \iff \mathsf{isresp\ (resp\ R,\ A)} \wedge \mathsf{isresp\ (resp\ R,\ B)}$$
$$\forall\ \mathsf{A\ transfer\ (resp\ R,\ B)} \iff \neg\ \mathsf{isresp\ (resp\ R,\ A)} \wedge \mathsf{isresp\ (resp\ R,\ B)}$$

levels. Even policies, only resulting from technical aspects, have at least one default subject, e. g., the IT officer, as the responsible subject.

For all these reasons, in this thesis the delegation process is considered as part of the processes involved in the policy-based management. It covers the organisational aspects which are necessary in a company.

4.2.4 Refinement Process

Insofar as policy refinement is concerned, only brief mention was made in [MoSl 93b, Wies 95, Koch 97, Heil 98]. Refinement of policies is usually necessary, because they are used to achieve a management objective, which is almost always formulated on an abstract level.

The refinement process is the most critical process in the application of policies for management, because it must be ensured that the semantics of the high-level policy is not changed. The initial high-level policy describing the principle is already formulated by the delegation process.

In the refinement process, the policy has to become more concrete through splitting it into several lower-level policies, which affect the same domain of subjects and targets. Alternatively it can be split into several lower level policies which partition these domains. Besides having different enforcement contexts (e. g., affecting other domains), these policies define (sub-)goals of their own. Together, they attempt to fulfil the high-level objective.

The following outlines some points to show the complexity of the refinement process:

- Relationships are introduced between newly created policies such as sequence, concurrency, or exclusion. The side effects relating to other policies can cause problems, because it is hard to predict what these effects are and how they might change the semantics of a high-level policy. Besides that, even ensuring that the lower level policies fulfil the high-level policy is very difficult. The reason for that is that refinement necessarily requires human intervention, and each person often has a different understanding of the original intention. The result might therefore be ambiguous.

- Refinement of policies also depends on external constraints like costs and efficiency, which also leave room for interpretation. For example, a high-level policy could be: "Only authorised persons have access to IT-systems." In this case several refinements are possible. Take the word 'authorised'. It implies the need to prove a person's identity through authentication. Authentication can be done in several ways, for instance with the help of biometrical methods, with a smart card, or a login-password-combination. Which technique to choose depends on external constraints, e. g., costs, availability, legislation, common practice, and/or other policies governing authentication.

- Because of the complexity, the refinement process will probably be repeated several times before it is completed successfully. During this period, it is not unlikely that a high-level policy has to be changed because the refinement reveals it to be wrong, ambiguous, or unenforceable.

- The refinement process must be well defined and documented to ensure that it is possible to review the decisions taken. This requires that the initial development of a policy, all changes, and the reasons for them are stored within the management system.

- During the refinement process, the policy gets more concrete, in that variables in constraints will be bound to objects, or to concepts such as roles. In this process, an object can possibly be further refined with the necessity to split an object into two parts. Thus, the more abstract domain of the former object is not bound any longer by the lowest level policies, which will be deployed. The existence of the initial refined object or some of its properties have implied the reason for the further refinement. In this case, a change or the removal of the initial higher level object can lead to a false or obsolete lower level policy.

As mentioned above, the refinement process creates new policy descriptions, or policy objects. Despite of this, it also uses existing policies for further refinement or for taking decisions during the refinement phase. This is depicted in figure 4.1.

4.2.5 Distribution Process

The distribution process in a policy-based system is necessary for spreading the policies to the places of their actual enforcement. Figure 4.1 shows the distribution process getting policy descriptions or objects from the global policy repository. Only deployable (thus refined low-level policies) are multiplied and distributed to the local policy repositories where new policy objects are created if the distributed policy was not already contained there. The decision of the proper destinations is taken by looking at the subjects and targets of a policy.

The distribution process has to consider whether there are constraints regarding the distribution. For instance, a policy must be distributed and stored in the local repository before another one. This could be the case when policy chaining is used as described in section 2.3.

Also duty-bound to the distribution process is guarantee that the policies transfered to the local repositories are provided with all the components and objects needed by the new policy. A component can be the monitoring part of a policy as suggested by [Wies 95] and required objects can be special objects the policy references to.

Some (automatic) customising may be in the responsibility of the distribution process, such as translating configuration files, or changing the representation of values (e. g., convert big-endian to little-endian) if not done transparently by the transport system or any middle-ware.

After the distribution process has completed its work, the policy in the local policy repositories will not be significantly altered. Small modifications, which do not change the semantics of the policy, are done by the test process and the enforcement process.

4.2.6 Test Process

When the deployable policies are in place at the local policy repositories, they will be tested. This process is necessary because dynamic runtime-constraints may be overlooked in the refinement process. The already existing policies at the local repository are in most cases also not considered during the development and refinement of a high-level policy. Even the compliance with a (technical) standard (e. g., ISO 900x) or law may entail detailed testing and logging of the results.

The test process may retrieve policies from other local policy repositories, as shown in figure 4.1, for instance when there is a chance that the new policy interferes with them.

As mentioned at the beginning of this section, it is not necessary that each process is implemented by a separate computer process. So, the test process may use functionality of the enforcement process, or be actually the same component as the enforcement process. In this case, the enforcement process would just work in a testing mode, which is switched to real enforcement when the testing is done. When a policy has passed all tests, it is marked as "ready" for the enforcement. The policy can now be activated by the enforcement process.

When the test process encounters a problem which restricts the enforcement of the new policy, the test process either notifies the refinement process, or the delegation process, or both. This depends on the local practice and on the problem. The notified process has to take actions to solve the problems occurred. Until then the policy will not pass the test process.

At present little is known about testing and ways to efficiently carrying out tests. It seems that the test process must be highly customised for different sets of policies, or in worst cases for every policy in a particular environment. This may be necessary, because a policy can be integrated in various kinds of (dynamic) environments.

4.2.7 Enforcement Process

The first step of the enforcement process with a new policy is to activate it. For this purpose the policy is marked as active, enabling other processes such as test processes to check the state of the policy. At the same time, the new policy will be enforced by the enforcement process whenever there is a reason for it.

Similar to the test process, a problem can occur during the enforcement. However, when an enforcement occurs, the managed object or system is likely to be in an impermissible state. For this reason, enforcement problems are more critical. Similar to the test process, the enforcement process notifies either the refinement process, or the delegation process, or both. It must be decided what to do with the policy causing the problem, such as re-enforcing, stopping enforcement on the particular target which caused the problem, or deactivating the policy.

4.3 Enhanced Policy Life Cycle

As can be seen in chapter 2 by looking at the examples of the different policy approaches, each policy passes a number of similar steps. In section 4.2, the processes which take part in these steps were generally described. Different policy approaches handle some of the functionality of the supporting processes in a different fashion, or a policy-based management system may simply omit some, such as the test process when it is not possible to integrate it. A policy life cycle is, therefore, restricted by the management system.

Because of these reasons, an *enhanced policy life cycle* is presented here. It can be seen as best practice for a policy-based management system, and it is not contrary to any of the previously presented policy approaches. It shows the steps when a policy is created, modified, transformed, enforced, etc.

The enhanced policy life cycle considers all the aspects of the previously presented life cycles, and introduces new aspects such as testing and conflicts, showing all possible transitions from a state, and which processes are related to these states.

In the following we give an overview of the enhanced policy life cycle as shown in figure 4.2:

refinement In this phase, the initial high-level policy describing the principle is already formulated. With this description, a decision must be made with respect to two areas: to whom the policy will be delegated for refinement, and what aspects of the policy are to be refined.

Even after a policy has passed this refinement state, it may return to this state from any other state because of testing problems or enforcement failures, or for the purpose of auditing. For reasons of clarity, these edges have been omitted in the graph.

When the refinement process is finished, the high-level policy should be deployable as a set of low level policies.

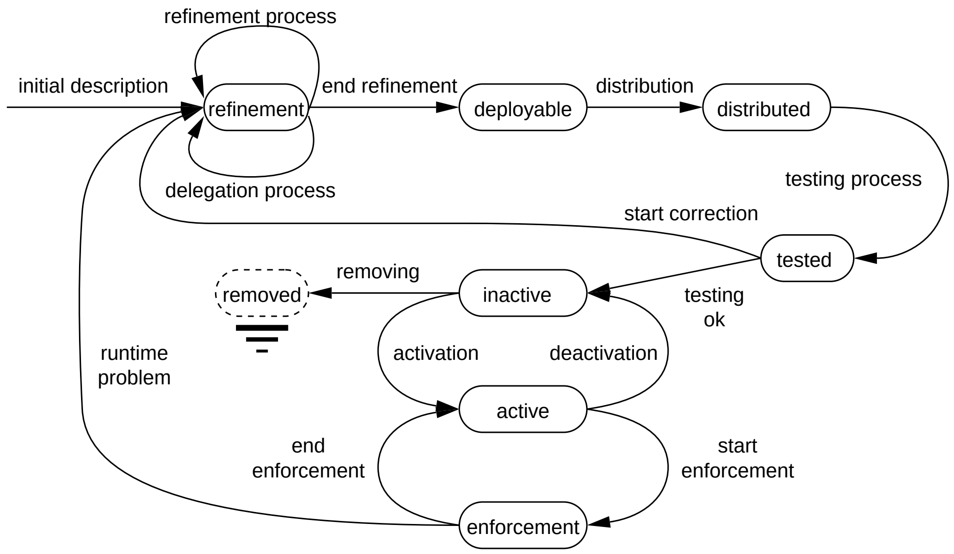

Figure 4.2: The enhanced policy life cycle

deployable The policy is fully refined and ready to use in this state. All objects in the policy can be mapped to the management system. As mentioned in subsection 4.2.5, some modifications, or customising for the policy destination may be made by the distribution process, but these changes must not change the semantics of the deployable policies.

The policies, therefore, can be distributed and, when they arrive at their destination, enforced. The policies in the entire refinement process are stored in a repository for the purpose of reuse and review, and would be retained until explicitly deleted. The next step is to integrate the policy, i. e. its low-level policies, into the system.

distributed For integration purposes, policies are distributed and stored in the local policy repository. During the distribution process, the destination systems are selected for each policy. The process is finished after the whole set of low level policies refined from a high-level policy is distributed and stored in an appropriate repository. The distribution process has to be completed before entering the test phase because with potential dependencies between policies, each policy will be required to be in place in order to be tested.

tested The distributed policies are tested to show their conformity with the initial intention as far as possible. This is done at this phase of the life cycle, where the system has access to the real world and can instantiate objects in the policy description and simulate them in a running system. The testing process depends on the policy; i. e., different policies can have different testing processes, which themselves can vary because of different environments/contexts in which the policies are enforced. Here, conflicts or other run time constraints, which lead to unwanted behaviour, can be discovered through simulating the enforcement of the policy. Activation and deactivation strategies are also tested, to detect possible problems in the integration process. These problems force one or more policies to return to the refinement state where the problems must be resolved. Otherwise, the test ends and the policy becomes "ready", or inactive.

inactive In this state, a policy is inactive, meaning it is integrated in the system and has no effect on the system, but is ready to become active through the activation process or to be removed from this particular system. A policy in this state has already been distributed to a hosting system, so the removal of a policy affects only this system and does not necessarily mean that the policy is deleted from the whole management system.

active Having activated the policy enables the system to enforce the policy, when there is an event which concerns the policy.

enforcement At this step the policy is finally enforced triggered by an appropriate event. Normally, after enforcement the policy goes back the active state again. If a conflict with another policy occurs, or an unforeseen exception happens, the policy will have to go the refinement state in which to get the problem resolved.

Review of policies is possible in and from every state. This is not shown in the diagram for the purpose of clarity. Reviewing as part of the refinement and delegation processes is in most cases a parallel process to the normal states in the life cycle. Actions initiated by the review need to conform with the policy life cycle. When a review leads to the conclusion that a policy has to change, then necessary actions are taken.

For example, if a new policy instance is created incorporating the changes made, then it must be (re)distributed. Even when only minor alterations are made, the policy must pass through all states in the life cycle to ensure its orderly (re)integration. On the other side, all instances of the previous version of the policy should be deactivated and then removed from all systems.

4.4 Summary and Conclusion

The literature has paid little attention to the intensive investigation of a policy life cycle and the processes involved.

This chapter presents an overview of the functional areas required by the application of policies. In contrast to existing publications, organisational aspects such as responsibilities, transfer of work and testing of policies are considered. This is necessary, because policies are not used in an encapsulated environment. Despite of this fact, the presentation always focuses on the policies themselves.

The functional areas are named and described, not exhaustively, but detailed enough to get an idea of their complexity. Each functional area is represented by a policy supporting process, which encapuslates the implementation of one functional area. Characteristics common to almost all processes are described by comparison with active policies. A policy centric functional model of the processes is presented and after that each of the processes were described.

An enhanced policy life cycle is introduced and it is showed that a policy passes certain distinctive states, and one or more processes or actions are related to each state.

In chapter 3, it is concluded that the application of policies in various scenarios needs additional information and control. This enhanced policy life cycle and the functional model show that the flexibility of policies potentially involve complex processes for policy-based management. This enlarges the requirements for additional information and control to the policy-based management itself. With the description of the supporting processes, it is concluded that policies and the functional units need control, information, and configuration. Understanding the processes as flexible functional units, like tasks in a workflow management system, entails constraints on these tasks and roles affiliated with them [BFA 97]. Therefore, management for the processes in the application of policies is necessary. This is seen as policy management.

In the next chapter, the concept of metapolicies is reconsidered. It will be shown that the the concept of metapolicies fulfils the requirements of providing information and control of the policy life cycle to support the processes of the life cycle.

Chapter 5

Policy Management with Metapolicies

This chapter introduces metapolicies. I give a definition of metapolicies I developed, resulting from the requirements for managing the entire policy moving process. Active and passive metapolicies are discussed, and a classification of metapolicies worked out by me is presented. Examples for each class of metapolicy is given and properties of metapolicies are discussed.

Not considered are here metapolicies for exchanging policies as Hosmer and Kühnhauser have suggested, because very little is known about their requirements.

5.1 Requirements for Metapolicies

Policies are seen as advantageous for management. They have been chosen for management of various areas in network and systems management. Management includes, among other areas, the management of performance, configuration, security, and fault. In this section, we show that these areas are included in the developed enhanced policy life cycle and its elaborated related processes. Consequently, we (re-)apply the policy-based management approach leading us to the definition of *metapolicies*.

Configuration The overall process of moving policies is complex, not only because different processes work in an interconnected fashion. The characteristics of the processes responsible for moving policies, as described in subsection 4.2, show some of the configuration needs: The processes use different sources of input. The processed information has different formats. The processes exchange and share all kinds of data, e. g. policies, notifications, responsibilities, etc. It is important, that the processes are configured consistently. A unified and central configuration management eases this task tremendously.

Performance Performance management is strongly connected to ensuring and raising efficiency. This is an important task in network and systems management as it is for the management of the policy supporting processes.

Only by a detailed description of the workflow, a policy has some degree of guarantee that it will move through the appropriate states of the life cycle and become enforced. Guidance and management activities are necessary to ensure this in an efficient manner.

Fault Various kinds of errors occur in the processes related to the policy life cycle: policy conflicts, enforcement and distribution problems, etc. Their detection and strategies for resolving them are tasks of management.

Security Ensuring a secure operation and preventing fraud entails management of the system as of the supporting processes. The specification of access rights and responsibilities are tasks ensuring a secure operation. Although security concerns are valid in the supporting processes and must be managed, they are mentioned occasionally but are not considered in depth in this thesis.

5.2 Definition of Metapolicies

A look at the policy life cycle and its related processes shows that there is a need to manage the policy system. There are in general, two different kinds of objects to control: policies and the processes involved in the policy life cycle. The supporting processes named in chapter 4 are refinement, delegation, distribution, testing, and enforcement. Reusing a policy-based management approach leads us to the definition of *metapolicies.*

> *Policies governing the phases of the policy life cycle and policies having other policies or policy supporting processes as targets are defined as metapolicies. Their statement is driven by management needs.*

In the first case, the meaning of *meta* is self-evident: policies about policies. In the latter, the term *metapolicies* relates to the processes of the policy life cycle. It is the aim to manage these processes with metapolicies using the same infrastructure, processes, and management systems used for normal management policies.

The relationships of metapolicies and their integration into a policy-based management system is depicted in figure 5.1. The metapolicies are derived from the basic policy concept. As in the case of normal policies, there are active and passive metapolicies. The metapolicies are used to managed the

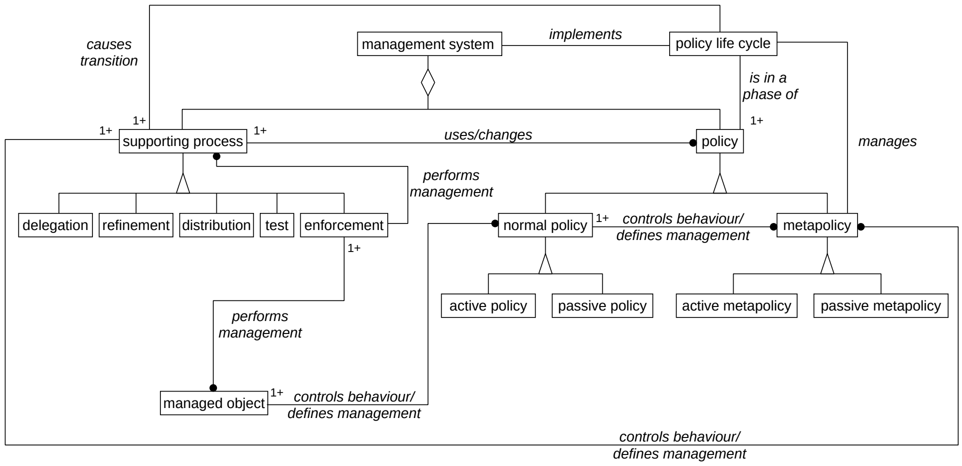

Figure 5.1: Relationships and integration of metapolicies

entire policy life cycle of policies. This includes controlling the behaviour of the policy supporting processes and defining management activities on them. This affects normal policies through their moving processes, but stating constraints directly on policies is also possible. Metapolicies consequently have other policies or supporting processes as targets. The enforcement process performs activities on objects of the management system on behalf of metapolicies. These objects have the role of managed objects in this case.

5.3 Active and Passive Metapolicies

Metapolicies share most of their properties with normal policies. Consequently, there are also passive and active metapolicies.

The state space metapolicies care about is given by the whole number of policies, the supporting processes as named in chapter 4, and their properties.

Passive metapolicies specify the permissible state space including valid state transitions. Active metapolicies react on perturbations of the permissible state space. With their help, a system becomes compliant with passive policies again.

In some cases it is possible to achieve the management tasks with either passive or active metapolicies. A reason for an explicit distinction may not be obvious immediately. Metapolicies assist in the management of policies and their related processes. Management tasks must generate as few expenses as possible, because management on its own offers no real value. Therefore, guiding the processes with additional information causes just minimal extra effort and is the preferable solution. This additional information source is provided by passive metapolicies. For this reason an explicit distinction of active and passive metapolicies also makes sense when the management task can be achieved by either one of them. In cases where only a passive or an active metapolicy is possible, it makes explicit the fact that the active ones cause state transitions by performing management activities.

Although it is important to distinguish between passive and active metapolicies, it depends on the actual implementation whether an active or passive metapolicy can be used. If a supporting process does not consider a stated passive policy, an (additional) active metapolicy must be used to force the process to comply with the intended behaviour. In this way, active and passive metapolicies work closely together. A real example can be seen in section 6.3.3, where limited attributes describing the state of a policy exist. A passive metapolicy describing the valid state transition of a policy says: "The state of an altered policy changes to modified." A refinement process causing the alteration of the policy acts on behalf of this passive metapolicy and marks the altered policy as modified. It acts while not leaving the permissible state space. In another system, the process which causes the alteration does not consider the stated passive policy. In this case, an active metapolicy must ensure that the state of the policy does change. This can be realised by triggering/invoking actions on the process. Whenever this is not possible, active policies must use other solutions, for instance as described in chapter 6.

In subsection 4.2.1, the differences between active policies and processes are described. Almost all of the different characteristics can be applied to metapolicies. One exception is the consideration of organisational aspects. This is an area where active metapolicies are not as different from processes as active policies are. Because, they consider organisational aspects as they manage for instance the delegation process. On the basis of the remaining differences it is clear that active metapolicies are no process descriptions. They assist and control processes, but do not specify complex tasks necessary to move policies. Consequently, active metapolicies do not aim at describing large algorithms. They just perform very specific management activities.

5.4 Classification of Metapolicies

In this section, a developed classification for areas of metapolicies is introduced and presented.

Using policies for the purpose of policy management often begs the question of how they can be brought into a well-understood structure. This is essential for many reasons, such as locating, reviewing and reusing a policy, or understanding the side effects during a change and test phase. Also, people who are affected by a policy should have the chance to see and understand the reason for this policy which is important for a high acceptance rate or in order to get feedback [Goh 98].

In the following, we will describe aspects that metapolicies must cover and classify them into management areas. These areas have a strong relationship to the states and supporting processes of the policy life cycle. The processes deal mostly with very different areas, so a classification related to the policy life cycle and the supporting processes provide metapolicies with a good structure enabling the writer and user to have a straightforward mapping from a metapolicy area to the affected parts of the policy life cycle.

5.4.1 Refinement

Any policy referring to the refinement or review process is a metapolicy which fits into this category. This category is different from 'Delegation', because it covers all aspects of the real refinement process. Delegation is much more focused on administrative aspects. An example for a refinement metapolicy is: "When a property of a domain has changed, all policies related to that domain must be reviewed."

5.4.2 Delegation

This category includes metapolicies which govern the delegation of policies. Here, delegation involves transfer and setting of responsibility and other business and administration related concerns. This is necessary to avoid ambiguities later in the policy life cycle. Therefore, it must be stated who has to deal with refinement, review, testing, and conflicts. Example: A security related policy must be reviewed by a security expert.

5.4.3 Integration

Metapolicies affecting distribution, testing, activation and deactivation are covered by this category. The constraints between policies have to be considered and adequate metapolicies must be formulated to ensure an orderly updating or the introduction of new policies. Example: Specification of the sequence of activation or distribution.

5.4.4 Conflict

Systems with a large number of policies usually suffer from having conflicts between different policies. In this context two things are important: detecting a conflict and resolving it. Sometimes additional information is necessary to detect a conflict. Metapolicies can state constraints over policies in order to specify either acceptable [LuSl 97d, LuSl 97a, LuSl 97b] or unacceptable cases for the application of policies. A strategy for resolving such conflicts can be implemented through metapolicies, with the advantage that the resolution is no longer tied to one specific approach. The resolution strategy for each policy, and respectively for each conflict, can be chosen independently.

It is very important that conflicts are handled in one place and in an orderly way. This is an imminent requirement for management.

5.4.5 Exception

Another problem with policies is, just like with any rule in the world, that at a certain moment, exceptions have to be made. They are usually transitional and get incorporated into the mainstream policy itself following a feedback process. On the other hand temporary policies are necessary for stating exceptions to the normal operation. For example: maintenance or changes of a system during normal operation in the IT environment will affect performance policies, because the required quality of service in this area cannot be fulfilled. So the performance policies will take actions or at least generate error events. This management system behaviour is undesirable, because the IT system operates orderly in terms of the maintenance situation. Metapolicies will handle this, because they can control other policies, and for that reason it is easy to create exceptions to prevent the management system from taking unwanted actions.

Policies governing the process of introducing exceptions into the system are also important. For example, who is allowed to sign off exceptions for a certain area or how long will the exception last? The advantage of collecting them in a separate class is that exceptions are explicitly stated and not treated as normal policies. If it is necessary to permanently incorporate the exception into the system, it must be considered at the refinement stage so that it also becomes a policy governing refinement.

5.4.6 Enforcement

All policies for enforcing relevant constraints, both on the policies and on the policy enforcement process, fall into this area, e. g., ensuring that the monitoring component is working before policies are activated.

5.5 Metapolicy Examples

In this section, some practical sample metapolicies to each of the above classification criteria are presented.

The metapolicies are stated in textual form because this does not restrict their investigation. This form shows some problems occurring during the development and refinement of metapolicies and, more importantly, it leaves space for the discussion of different implementations.

5.5.1 Refinement

Metapolicies in this category govern the development and the refinement of policies, thus influencing the processes or (human) development procedures involved. Table 5.1 shows example metapolicies:

1. Only a child policy may have a parent policy.
2. Every child policy must have a parent policy.
3. A modified policy must be reviewed, before being activated again.
4. If a policy is modified all child policies must be marked as modified.

Table 5.1: Example refinement metapolicies

The examples will be discussed briefly and requirements for their application and enforcement will be investigated:

- The first two metapolicies ensure consistency in the policy structure in the sense that a policy which is a child policy must have a parent policy. If it is not a child policy it is not allowed to have a parent.

  This policy is a passive metapolicy, because no explicit actions are stated and it describes the desirable state space of a policy. It is necessary that the system can distinguish between a child and a parent policy, but it causes no problems if a policy is both, as one in the middle of a policy tree, for example.

- The aim of the third metapolicy is to trigger the reviewing process if a policy is modified. This is necessary, for instance, because unwanted management operation caused by a policy which is discovered to be wrong may do harm in large parts of the managed system. Reviewing before its enforcement reduces the probability of unwanted effects. Many other reasons exist such as checking its conformance with parent policies and check whether other policies refined from the actual one are consistent with the modification.

  This example metapolicy is an active metapolicy, because an action is specified which must be executed when a perturbation of the desirable state space is noticed. (Here, it is the (re)activation of a modified policy.) The management system must provide the information whether a policy is modified or not. Different techniques for this are discussed in chapter 6.

- The fourth metapolicy is an active policy and is used to ensure that the child policies are consistent with the parent policy. This is ensured in combination with the third example metapolicy.

5.5.2 Delegation

Metapolicies in this category govern the delegation and other organisational aspects related to policies and the supporting processes. Table 5.2 shows example metapolicies being part of the delegation metapolicy classification:

1. Only the creator of a policy can authorise changes of that policy.
2. The creator and a second person must review the policies.
3. Delegation of refinement work does not transfer responsibilities.

Table 5.2: Example delegation metapolicies

- The first metapolicy clarifies who has the power to allow changes of a policy. This may be necessary if the creator is responsible for a policy created by himself. Only the creator having the power to authorise changes ensures that he must be aware of changes and that he can take appropriate steps (e. g. not authorise the change, when he does not take the responsibility for that change).
- In the second metapolicy, constraints about the review process are specified. This information is considered when reviewing policies. The specification of the review process in its entirety is not made up by metapolicies, but information governing it definitely is.
- The third metapolicy specifies a general principle saying that transfer of workload does not include the transfer of the responsibility.

5.5.3 Integration

This class of metapolicies governs the distribution and test concerns. They are used by the distribution process and by the test process. Table 5.3 shows some example metapolicies for the integration process:

- The first metapolicy specifies a general constraint in the integration phase. The management agent may be responsible for registering or testing a policy in its domain. For this, it must be running correctly in the integration process.
- In the second metapolicy, an additional constraint is added to a policy state. This may force the system keep the policy persistently stored or trigger an action resulting in the transition of the policy to an inactive state.

1. The agent managing the target object domain stated in a policy must run during testing and before activating the policy.
2. An inactive policy must be stored persistently.
3. Windows NT systems must be rebooted before an integrated policy can be activated.
4. A policy is deployable after it has been reviewed and its formal policy description has been syntactically checked.

Table 5.3: Example integration metapolicies

- The third example specifies a constraint for the activation of an already integrated policy. A successful enforcement can have external conditions that must be satisfied in advance. This has to be ensured before a policy becomes active.
- The last of the integration metapolicies governs explicitly the state of a policy in the policy life cycle. This has consequences for other supporting processes such as the distribution process if it is not allowed to start the distribution before a policy has reached this state.

5.5.4 Conflict

The example metapolicies in this subsection show the flexibility of handling conflicts. They are often related to more than one supporting process. In table 5.4, some example metapolicies in this area are given:

1. If several agents are managing the same target with the same policy, only one performs management actions.
2. A policy is in conflict if another policy with the same subject, target, and constraint expression exists.
3. A policy is not valid if it states management actions performed on the management system.
4. A not existing subject of a policy makes this policy inactive and invalid. It has to be reviewed.
5. Policy `X` and policy `Y` are in conflict when enforced concurrently on the same target.

Table 5.4: Example conflict metapolicies

- The first and second example metapolicies provide data which allows the detection and the resolution of conflicts. This information can be used during the test and enforcement phase as in the refinement process to avoid conflicts in advance. The examples forbid the triggering of the same management actions on one object. This shows that nearly the same semantics is expressed with two different metapolicies. One focuses on the specification of conflicts, the other on a strategy for conflict avoidance. It may not be a desired behaviour that several policies are triggered on the same object. This can be a reason for management operations being performed several times, which is not acceptable if the operations are not idempotent. Or, in the case of different actions a "ping pong" configuration may be the consequence. A solution is to trigger one policy, which in turn triggers other necessary policies in a chain.

- The third metapolicy denies normal policies to control the management system. This makes only sense when normal policies and metapolicies can be differentiated. This metapolicy is necessary because metapolicies and normal policies share the same infrastructure. In a metapolicy-enabled system, controlling the management system is done with the help of the same technique normal policies uses. As a result of this, normal policies can invoke management actions on the management system. This is not wanted and must be denied for security, vulnerability, and clarity.

- In the fourth metapolicy, a general constraint on a policy is specified. This is as a metapolicy of the conflict class, because it ensures that it is considered in all phases of the policy life cycle.

- The last metapolicy is for two specific policy instances. It denies concurrence of enforcement on the same target. In contrast to the first and the second example, the subject can be different. This policy may be needed because inconsistencies can arise as a result of concurrent enforcement.

5.5.5 Exception

The following examples in figure 5.5 specify some exceptions for a policy-based management environment in place.

- There are situations that are exceptional in comparison to the normal operation. These special situations may entail exceptions for the operation of the supporting processes or the enforcement of normal policies. The first metapolicy does not allow putting any new policies in place, because this could falsify the auditing results.

1. During policy auditing, there is no integration of new policies.
2. It is not possible to activate policy `X`.
3. Stop policy enforcement in domain `NT`.

Table 5.5: Example exception metapolicies

- Due to certain circumstances, exceptions for a specific policy or for policies on a specific target (domain) must be made. This can be realised as shown in the second and third metapolicy examples. The advantages of integrating such exceptions in metapolicies are the explicit statement and central management of the exceptions.

5.5.6 Enforcement

The policy enforcing components need additional information affecting the enforcement of all or a specific subset of policies. A few examples are given in figure 5.6.

1. Security agents must be authenticated to ensure trusted events.
2. Security events must be sent encrypted.
3. Policies in conflict must not be enforced.
4. Policies which must be reviewed are not enforced.

Table 5.6: Example enforcement metapolicies

- In cases where the origin of events is crucial for the triggering of management actions, extra constraints may be imposed on the sending component. The first metapolicy ensures only the processing of security events received from authenticated agents. Additionally, when the content must not be made public, the agent is obliged by the second metapolicy to send the events encrypted.

- The enforcement processes are given a rule how to handle policies found being in conflict, e. g. through conflict metapolicies. Also possible are guidelines for policies having a specific state attribute. The last two examples show such metapolicies.

5.6 Properties of Metapolicies

Metapolicies require the greatest expressive ability in a policy-based management system. They combine the properties of normal policies with the concept of self-referencing and thus self-management. A policy controlling one or more regular policies is a metapolicy. A policy controlling some state in the policy life cycle or some supporting processes is also a metapolicy. Therefore, metapolicies are used for managing regular policies and their supporting processes.

It is also possible that metapolicies are only used to govern special areas of interest as presented in section 5.4. In case it becomes necessary to expand the application area of metapolicies, they are powerful enough to express the needed concerns, if the management system provides them with the needed API. Another possibility would be to support only passive metapolicies at first and active policies (where the API is needed) at a later stage or vice versa. This shows that the support for metapolicies can scale.

With the support and application of metapolicies critical situations can arise. These are the consequence of the dual roles of the management system and its self-management. For example, it is very easy to lock up the management system with a metapolicy: A metapolicy states that a component must wait for the reception of an event before it begins work. But the component which must send the event is also affected by this metapolicy. As a consequence the system is deadlocked and the metapolicy has created a conflict. This is a result of the dual roles of the management system where the target and subject can be the same object. As a consequence a subjects-targets-overlap [MoSl 93a] can happen.

If metapolicies are needed for normal management, either some constraints have been left out during the normal policy refinement process or the environment has changed. In both cases the policy must be reviewed and changed, rather than using metapolicies as a 'quick and dirty' fix. For example, policy `A` governs a Unix domain, but from a certain moment on only Linux systems in the Unix domain must be governed by policy `A`. Deleting the non-Linux members from the Unix domain is not a good idea, because other policies may apply to them. A metapolicy can solve the problem: policy A is only applied to the Linux systems of the Unix domain. As mentioned in the 'Exception' criteria, through this mechanism the fix is explicit and will not be forgotten for any reason. This supports the aim of management and creates no additional problems, because IT management in general is a supporting process and must be as clear and concise as possible.

Metapolicies have the property of self-application, because of their definition. An example is the setting of the responsibility for metapolicies. This is useful, but this is also an argument for restricting the authority to use metapolicies more than for normal policies.

Metapolicies are policies. For this reason, they inherit the same properties and the same life cycle. However, differences exist. These differences are not structural but have to do with the quality of the processes. Examination of examples has shown that the refinement process is less complex, because metapolicies do not have to deal with so diverse an environment as normal policies. A top level metapolicy could be: "All policies must be signed off by an officer with authority." The refinement would be: "Security policies must be signed off by a security expert. Availability by the customer support manager, performance by the system manager", etc. These are implementable metapolicies, even though they seem to be very abstract. But with 'security policies' it is clear who must sign them off, because at this level 'security policies' is a class of objects, which can be instantiated, so no further refinement is necessary.

The development of metapolicies is, in some cases, parallel to normal policies, as it can be seen from the 'Integration' criteria. Is a constraint been noticed in the refinement process (e. g., concerning the activation), a metapolicy will be stated in the policy's refinement process.

As with policies, the metapolicies need an appropriate information and system model. This models all the aspects of the managed objects and provides the basis for processing information. The metapolicies inherit this model from normal policies, as they sometimes have to directly address the system resources, as seen in the 'Exception' criteria. Additionally, a model for the policy life cycle, the related support processes and its management must exist. Only with this model, the management system can manage itself using the same technique as for normal policies.

5.7 Benefits of Metapolicies

The most evident benefit of metapolicies is, that they fulfil the demand of policy management. The way they accomplish this is as flexible and sophisticated as policies are.

Reusing the same technique entails consequently reusing the same environment and infrastructure. An appropriate environment for policies must be in place in any case and reusing it for metapolicies causes only minimal extra effort and costs. And, expertise gained in the area of policies is easy to transfer metapolicies. No other management mechanism can integrate so well into a policy-base management system.

As mentioned above, metapolicies require the greatest expressiveness, but the area of their application is small in comparison to normal policies. In many cases, this results in straight forward refinement and integration procedures. However, this does not mean that handling metapolicies is easier in any case. Especially metapolicies of the conflict area require in depth understanding of the entire policy mechanism implemented by a system. Results on research of conflicting policies are still very limited [MoSl 93a, LuSl 97a].

It is another advantage, that metapolicies remain, once developed, to be quite stable in certain areas. This is a result of the properties of policies in general, because they are seen as statements of best practice [Wood 97]. It is true especially in those areas dealing with organisational aspects and the management system itself, because these areas stay more stable than the rest of the fast changing IT environment.

5.8 Summary

By comparing metapolicy supporting processes with systems management, we can derive, that policies and the policy life cycle need to be managed. Metapolicies are the appropriate concept fulfilling the management needs. A definition of metapolicies resulting of the self application of the policy concept is given. The difference between passive and active metapolicies is shown. A classification of metapolicies is elaborated. The classification presented covers all management areas needed to manage the policies, their life cycle and the supporting processes in a policy-based management system. This classification enables the writer of metapolicies to associate a metapolicy with a certain state or process in the policy life cycle. Examples for each class of metapolicies are presented. Properties and characteristics of metapolicies are discussed.

The next chapter shows the integration of the metapolicy concept into an existing policy-based management system.

Chapter 6

Policy-based Management System

In this chapter, an existing policy-based management system is presented serving as an example to describe necessary enhancements for the application of metapolicies. The requirements for metapolicy support are investigated. I will discuss how metapolicies depend on characteristics of a policy-based management system in general and which properties a system must support for the use of metapolicies in particular. I will give a model for a basic metapolicy implementation. The chapter ends with an evaluation of the scalability of the chosen design and gives some ideas for enhancing it.

6.1 An Example of a Management System

In this section, an example policy-based management system will be briefly presented. A full description of this system can be found in [Radi 98] (in German). From now on, we will call it *NoCScontrol*, which stands for *Nomadic Computer System Control*. It will be used to discuss

- the necessary enhancements for supporting a set of metapolicies,
- the application and use of metapolicies in a management system, and
- the limitations resulting from an existing management application.

6.1.1 Purpose

The management system was developed at the *Munich Network Management Team*[1] as the practical part of a diploma thesis [Radi 98]. It serves as a basis for future development and research in the area of network and systems

[1]URL: `http://wwwmnmteam.informatik.uni-muenchen.de`

management. Policy-based management is one area of interest. For this reason the management system supports policies from the beginning.

The NoCScontrol as described here was developed until August 1998 for supporting the configuration management of nomadic computer systems in an intranet.

6.1.2 Policy-based Approach

The policy approach implemented by the NoCScontrol system was mainly influenced by the policy work by Koch presented in section 2.3.

As in Koch's work, three different policy types reflecting the refinement of policies are used: Strategic policies, goal-oriented policies, and operational policies. Strategic and goal-oriented policy objects are only used as containers for the policy description supporting the refinement process. They are not enforced in any way. For this reason, the operational policies are the most important ones, because they are the most detailed ones and are enforceable by the management system.

Events

The specification of events is not separated from the policy description. Each policy has to introduce the event (name) and its structure the policy makes use of directly. Defining new events is not supported, because an event itself is received from agents and must be well known. Therefore, a separate event description language does not exist as in Koch's work.

The event specification as part of the policy description consists of the name of the event and its attributes following the keyword `DELIVERS`. Table 6.1 shows the structure of the policy description. The attributes are bound to variables which are typed with `string`, `long`, or `boolean`. With the help of the variables, it is possible to reuse attribute values in the specification in the `ACTION` and `CONSTRAINT` parts of the policy. An example is given in table 6.2.

As a result of this, there is no distinction between different kinds of events, like monitoring, timing, or composed events. Basic event correlation is also not implemented, but could probably be done in a decentralised fashion by the agents of the management objects or in the constraint part of the policy specification.

Policy Language

The *Policy Description Language (PDL)* used for the operational policies is also guided by Koch's Policy Definition Language and implements a subset of it. Table 6.1 shows the structure of an operational policy in PDL notation.

The PDL does not distinguish between authorisation and obligation policies. The policies stated with the PDL are on the one hand obligation poli-

```
POLICY name
FOR subject
ON <target>
DELIVERS event {string | long | boolean attribute}
ACTION {
 idl_operation() | subject.idl_operation() | object.idl_operation()}
CONSTRAINT (... && ... | ... || ...)
```

Table 6.1: Structure of the PDL notation [Radi 98]

```
POLICY SwitchCreateVLAN
FOR /libraries/
ON <NS:/libraries/our_library/private/sw1.pc_switch>
DELIVERS NewNodeAttached {string port,
                          string NewNodeHardwareAddress}
ACTION
{openPort (port);
 addToVLAN (/libraries/our:library/mail1.mail_hub/.get_VLAN_ID(),
 NewNodeAttached:port);}
CONSTRAINT (Subject.getState()=authenticated)
```

Table 6.2: Policy example in PDL [Radi 98]

cies, because they explicitly specify actions executed after the reception an event. On the other hand, the policies are used to configure the infrastructure for granting access of nomadic computer systems to special resources. Therefore, they can be seen as result of the refinement of authorisation policies.

As discussed in section 2.4.3, obligation policies have the characteristics of active policies, and authorisation policies have characteristics of passive policies. As a consequence, only active policies are directly implemented by the NoCScontrol system.

Generating events by enforcing a policy is not directly supported by the PDL. Koch uses this technique for policy chaining on success or failure of a policy.

A sample policy is given in table 6.2.

Policy Life Cycle

The main focus was to establish a flexible architecture for the enforcement of policies in the development of the NoCScontrol system. This resulted in a limited number of supporting processes at this stage of the implementation. As a result, the policy life cycle reflects mainly the enforcement of policies. The policy life cycle in the NoCScontrol system is shown in figure 6.1.

A supporting process exists to syntactically check the PDL description of a stated policy.

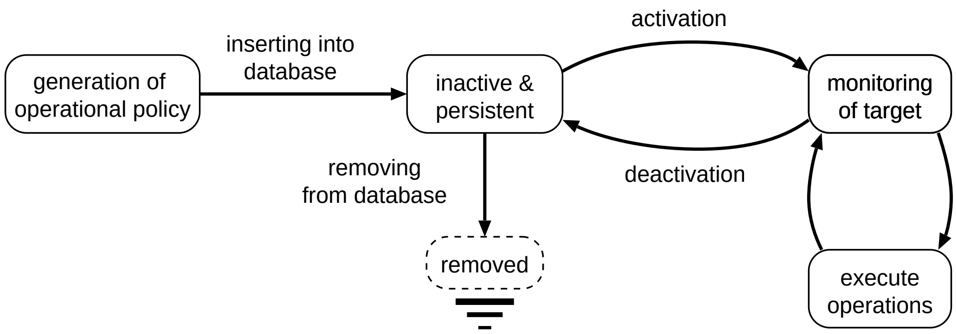

Figure 6.1: Policy life cycle of an operational policy in [Radi 98]

6.1.3 Architecture of the Management System

In this subsection, an overview of the architecture is given to get a conceptual view of the management system and its components. Not all components are described here for the purpose of clarity. Only the components relevant to this thesis are presented here.

The management system is realised by a number of distributed components made up of agents using CORBA functionality and services. All objects are CORBA objects and therefore can be referenced through the methods provided by CORBA. The objects can provide functionality with an IDL interface, which can be used from other components.

The architectural overview is shown in figure 6.2. The core system consists of the following components:

Agent System The mobile agents live on an agent system. There is one agent system per host. Mobile agents are created in an agent system which has the ability to control all of its agents. The mobile agents can migrate to other agent systems for the purpose of availability, load balancing, etc.

Mobile Agent A mobile agent is used to realise a special service on an agent system. The agent acts as a CORBA server and has a public interface to offer its service which can be used from other CORBA objects in the role of clients. As compared to a stationary agent, it can migrate to other agent systems.

Service A service provides functionality through a specific public interface. Here, a service is always made up of two components: a factory and the objects produced by that factory. Both the factory and the objects produced by it provide an interface to access their functionality[2]. Therefore, they act as CORBA server objects, but they can also use

[2]The only exception to this is the Persistence Service.

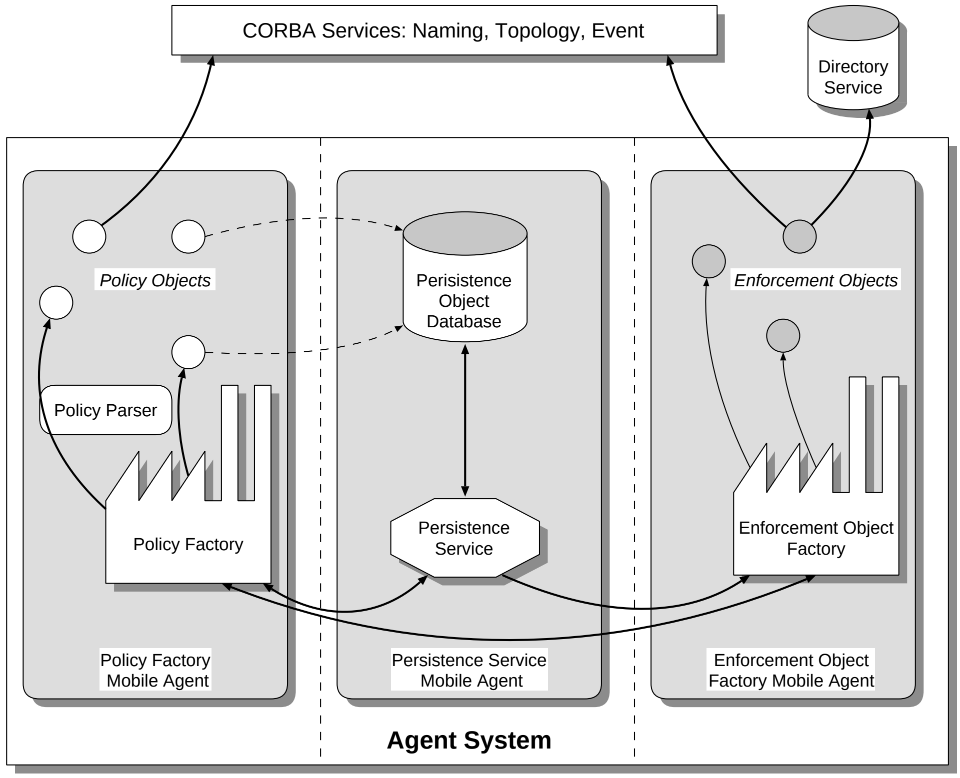

Figure 6.2: Architectural overview of the management system [Radi 98]

services of other server objects, acting as clients in this case. In the management system, the following services can be used:

Policy Service This service, which is represented on the left in the architectural overview in figure 6.2, is realised by the *Policy Factory Mobile Agent*. The Policy Factory Mobile Agent consists of the *Policy Factory* and the created *Policy Objects* by it. As can be seen in the more detailed figure 6.3, the policy factory is able to produce three kinds of policy objects: *Strategic Policy Objects*, *Goal-oriented Objects* and *Operational Policy Objects*.

In contrast to strategic and goal-oriented policies, an operational policy is described by a *Policy Description Language* (PDL), which can be checked to ensure its syntactical correctness. An operational policy, which is inactive at first, can be activated causing the interpretation of its policy description by the *Policy Interpreter*. If necessary, the interpreter invokes the creation of *Enforcement Objects* by using the Enforcement Service and binds the operational policy object to a special event channel, the *AgentInitChannel* by calling a method of the policy object for this pur-

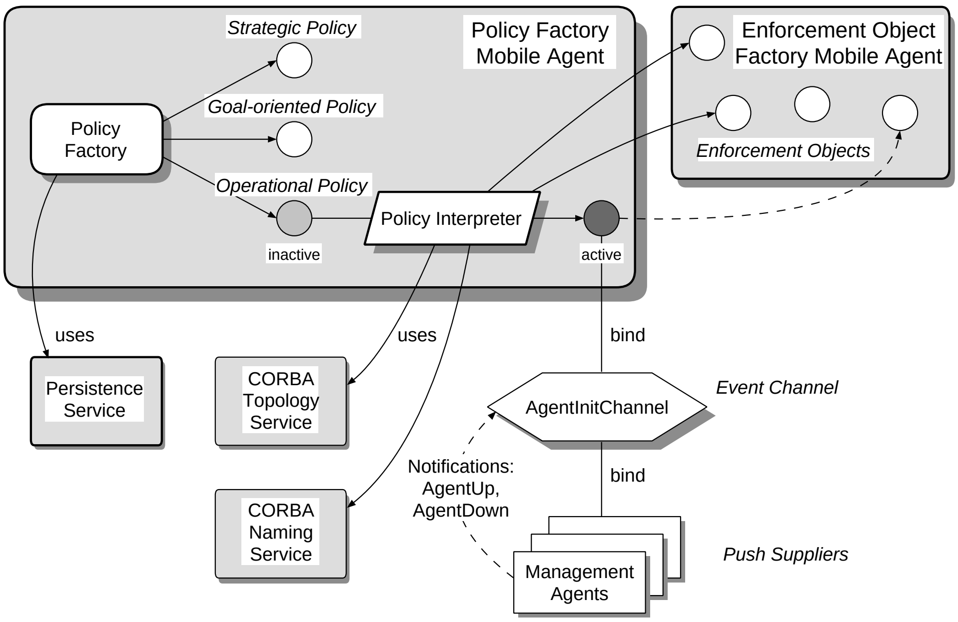

Figure 6.3: Details of the Policy Service [Radi 98]

pose. Through the AgentInitChannel the operational policy gets notified if new agents are added or removed.

For decoding the policy description and in order to reference the correct objects, the policy interpreter can make use of the CORBA Topology Service and the CORBA Naming Service. The persistent storage of policy objects is implemented by the Persistence Service.

Persistence Service The Persistence Service at the centre of figure 6.2 is realised by the *Persistence Service Mobile Agent*. It stores CORBA objects, such as created policy objects, and saves their actual state in a Persistence Object Database. It implements methods to store, restore, insert, and extract persistence objects.

Enforcement Service The right-hand side of figure 6.2 depicts the *Enforcement Object Mobile Agent*, which realises the Enforcement Service. As the Policy Service, it consists of two kinds of components, the *Enforcement Object Factory* and the *Enforcement Objects* created by it. An Enforcement Object Mobile Agent is always specific to a special type of management agent, for example a DHCP[3] agent, and can only create Enforcement Objects for this kind of agent.

The creation of Enforcement Objects is usually invoked by the

[3]Dynamic Host Configuration Protocol

Policy Service by activating an operational policy through the Policy Interpreter. Already created and activated policy objects which are stored in the Persistence Object Database also cause the creation of corresponding Enforcement Objects if the type of the Enforcement Object Mobile Agent matches the type specified in the policy as target.

Figure 6.4 shows the details of the Enforcement Service. After creation of an Enforcement Object, the monitoring of the target management agent can be started through binding the Enforcement Object to the event channel of the agent. If an Enforcement Object receives a notification then it starts to verify whether the enclosed subject of the event matches the subject of the policy the Enforcement Object represents. Because subjects are always nomadic computer systems, this is done with the help of the directory service. After that, the *Constraint Interpreter* evaluates the stated constraint in the PDL description of the policy and activates the *Action Interpreter* in case of a positive result. The Action Interpreter finally invokes the management actions specified in the policy at the target agent which was the creator of the notification.

Monitoring Service The Monitoring Service is implemented by using the CORBA Event Service. Monitoring is the responsibility of the agents for managed objects. They must use the push model variant of the CORBA Event Service and can send events to an event channel where all objects registered to this event channel get notified of the produced event.

Domain Service Two different kinds of services are used as domain service, on the one hand the CORBA Naming and Topology Services, on the other hand a directory service.

With the help of the CORBA Naming and Topology Services the structure of the network and its domains can be queried.

The directory service is used to query the home domain of a nomadic system. It is necessary to verify whether the home domain of the subject specified in the policy matches the home domain of the nomadic system identified by its MAC[4] address. At the moment, a relational database is used to match the MAC address with the home domain of a nomadic system [Schi 98].

[4]Medium Access Control – layer 2 in the ISO-OSI network reference model

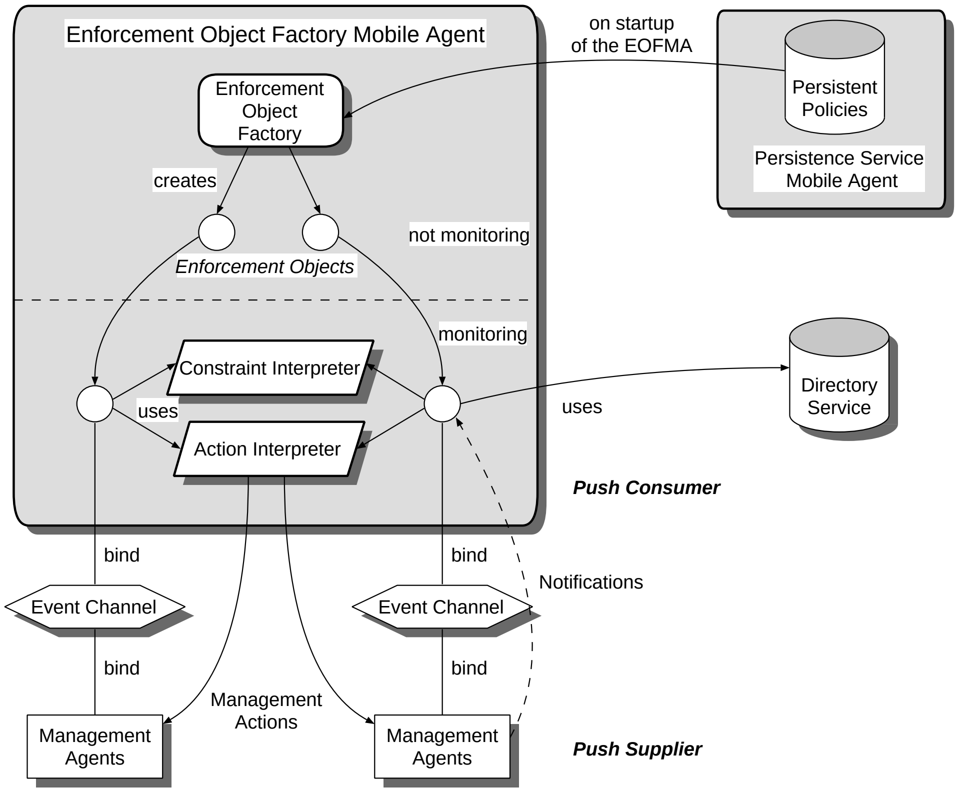

Figure 6.4: Details of the Enforcement Service [Radi 98]

6.2 Metapolicies and the Management System

In the first subsection, a description of the NoCScontrol system was given. In this section, I will briefly discuss general restrictions on metapolicies in a management system. I will introduce a categorisation of the areas of requirement. Based on these categories, I will suggest how the requirements can be met in principle. The next section uses some of these suggestions to model basic metapolicy support.

6.2.1 Restrictions on Metapolicies in a Management System

Metapolicies support the management needs of a policy-based management system. Therefore, it is clear that various management systems, which work with various concepts and implement different (management) features, have different problems and management concerns.

The metapolicy concept must be able to cope with these different requirements on the one hand. On the other hand, not all management systems allow the formulation of enforcement of useful metapolicies, as the following example shows: A management system which is not able to distinguish between roles like security expert, performance expert, etc. does not need metapolicies which have these roles as subject. General considerations regarding the restrictions of metapolicies in an existing management system are presented here:

- Usually, the flexibility and expressiveness for stating policies and metapolicies is given by the policy-based management system and/or the actual practice how policies are handled in the various stages, i. e. the policy life cycle. The first one implies a technical restriction, the latter does not. Despite this fact, changing a well-tried and established practice usually causes some trouble and therefore is often not wanted. For this reason, metapolicies should only be stated in a way which is compliant with the management system or the common practice of handling policies. Otherwise, as in the first case, the stating of metapolicies would not be possible or their enforcement would fail. In the second case, a well-tried and established practice would be destroyed in the long run.
- In addition to this, metapolicies, as policies in general, dependent on the usage of objects, their attributes, and relationships between them. If a management system is, for instance, not able to distinguish between a policy which is tested and a non-tested policy, it is not possible to state a metapolicy which handles these cases differently. This is a matter of the information model the policy-based management system supports and this model and the containing objects must be defined before a metapolicy uses them. Otherwise, it is not possible to reference such objects in the metapolicy. During the development of various metapolicies it became clear that the ability of a management system to add objects, attributes, and relations while specifying a metapolicy helps the metapolicy writer, because it is very hard to predict in advance which information and which objects are needed.
- A similar dependency of metapolicies exists in connection with events. Most management systems listen to events, correlate them, and take actions according to the changed situation. Known events are used with policies to trigger or restrict actions according to a policy. Metapolicies are no exception to this. The difference is, that in case of using metapolicies for managing the management system, it must change its role from consumer to producer of events. This enables metapolicies to govern the policy life cycle, because they can state constraints and actions on changes within the management system, i. e., its supporting processes or policies which are created, altered, distributed, etc. An example of a metapolicy which uses such events could be: Mark all child policies of policy `X` as "must be checked", if policy `X` is altered. The event which must be produced is the alteration of policy `X`. The metapolicy

states in the example how the management system must react with receiving this event.

- What can also be seen from the former example is that especially active metapolicies use actions which somehow must be transformed into procedures or methods of the management system. In the example, this is the marking of the child policies. These methods must be provided by the management system. Otherwise, the metapolicies cannot be enforced. Thus, the expressiveness of metapolicies directly depend on the available methods provided by the management system.

In this section, it was shown that metapolicies are dependent on the policy-based management system, and that the management system restricts the application of metapolicies. The more information and methods are provided by the management system the greater the expressiveness of metapolicies is.

6.2.2 Requirements for Using Metapolicies

In this subsection, the investigated requirements a management system must meet are listed comprehensively.

Passive and Active Metapolicies

As we have seen from the metapolicy examples and as was discussed in section 5.3, some metapolicies must be actively enforced and others, the passive metapolicies, state information which must be considered during the operation of the management system.

Both kinds of metapolicies are necessary, even when, in theory, the passive ones could be emulated by active metapolicies. However, they have different character and therefore the distinction is valid and useful.

The active metapolicies ensure the execution of actions and bring the management system into a state compliant with the metapolicy. With help of passive metapolicies, the management system can make right decisions without disturbing the desired management actions.

Managed Objects

Introducing the concept of metapolicies comes along with the introduction of new managed objects for usage in metapolicies. Metapolicies govern the behaviour of the management system and its supporting processes. As a consequence, the whole management system and all its components and processes are managed objects. This enables the management system to govern itself through metapolicies stated about theses objects.

State of Managed Objects

As a consequence of the introduction of new managed objects, the states of these objects must be observed. The information gathered serves as a basic source for the application of metapolicies. With the help of the observation of the state and state transitions of objects, information is risen which triggers active metapolicies or, in case of passive metapolicies, a decision is made whether an action, i. e. state transition, is valid.

Management Functionality

The management system must provide functionality to enable the metapolicies to specify the desired behaviour of the system with sufficient management operations. Especially the active metapolicies invoke actions on the managed objects, which are in their entirety the management system. Therefore, the system itself must have implemented the functionality to provide these actions for the enforcement of metapolicies. This is comparable to the management functions the management system invokes on behalf of policies on normal managed objects, because metapolicies use the same approach, but with different subjects and targets.

Policy Representation

Having satisfied all the requirements presented before, the metapolicies finally must be stated in an understandable fashion for the management system. Every management system must have a built-in mechanism which allows the formulation of policies. It must be investigated whether constraints exist which restrict the statement of metapolicies. The additional requirements to normal policies are:

Subjects The subject of a metapolicy is in most cases the management system itself or a subject specified through a role, for example the role of a security manager. Referencing to these subjects must be possible in the metapolicy representation and the enforcement process must be able to use the subjects.

Targets The targets of the metapolicies must be specified, as subjects are. Thus a method to specify them without ambiguity must exist. Often, the domain for the subjects is the same as the targets. So being able to reference a subject entails that the specification of targets is also possible with the same technique, for example domain expressions. The actions stated in the metapolicy act on the targets or are invoked by them, therefore the enforcement process must be able to execute the actions for the targets.

Events The representation of events must include the events received from the management components, which are added for metapolicy support. They are representing the state of the management system and the state transitions. Important is, therefore, the ability to specify the notifications when a management action is executed by the system.

Constraints The expressiveness, and thus the representation of constraints of metapolicies, must include expressions, i. e. for concurrency, sequence, running, not-running, etc. Metapolicies govern actions and processes. Therefore, it is necessary that during the enforcement of a metapolicy these facts are considered. This predicates can be simulated by low level events, but the ability to specify these concerns in a more abstract fashion would increase the readability.

Actions Various kinds of actions must be represented: In short, all actions necessary for changing the state of the system. Potentially these are used by metapolicies. In the specification of metapolicies it is required that actions can be invoked on different kinds of objects. This make the statement of more complex metapolicies possible.

Metapolicy Enforcement

Finally, the metapolicies must be enforced. Hereby, differences in which phase the metapolicies will be enforced exist between passive and active metapolicies.

Active metapolicies invoke specially stated actions to make changes on a managed object and are triggered by events which notify the perturbation of the system state. If the management system already supports enforcement of actions by receiving an event and if all the requirements described before are implemented, no additional changes for active metapolicy support should be required.

Passive metapolicies are guidelines for the operation of the management system. With the help of them, the management system makes provisions for not getting into an impermissible state. Consequently, the management system must consider passive metapolicies before it executes an operation. This will cause an alteration of the management system in order to consider the passive metapolicies.

6.2.3 Realisation of Metapolicy Support

In this subsection, suggestions for implementing metapolicy support into a management system are given. The NoCScontrol system will serve as an example management system. The necessary improvements needed to fulfil the requirements described in section 6.2.2 will be discussed.

Definition of ischild: $\mathsf{ischild(X)} \iff \exists\ \mathsf{Y: child(Y) = X}$
Invariant: $\forall\ \mathsf{Policy\ X: ischild(X)} \iff \mathsf{parent(X)} \neq \mathsf{null}$

Operation	Operation compliant if
delete (PolicyID Y)	child (PolicyID Y) = null
Y.addChildID (PolicyID X)	parent (PolicyID X) = Y
removeChildID (PolicyID X)	parent (PolicyID X) = null

Table 6.3: Metapolicy invariant and constraints on policy operations

Passive and Active Metapolicies

As already mentioned in section 6.1.2, obligation policies have similar characteristics as active policies. The NoCScontrol system only supports obligation policies, which means in order to support active metapolicies no conceptual improvements are necessary.

In the system, passive policies and metapolicies are not considered and implemented yet. The system must be enhanced to support them. With the help of a passive example metapolicy, it is shown what the enforcement of a passive (meta-)policy means for the state of the management system. As an example, a metapolicy is chosen from the refinement criteria: Every child policy must have a parent policy.

Passive Metapolicy as an Invariant Characteristic for a passive policy is the fact, that before and after execution of an operation the policy is valid. Thus it is an invariant of every operation in the management system. This high-level description states in particular an invariant of the policy structure. Transformed into constraints, the metapolicy would be noted as shown in table 6.3.

In table 6.3, some operations and the constraints they must fulfil after their execution are listed. A management system must be able to make a decision whether an operation is permitted or not. Therefore, the operations must be specified in a way which allows the system to make such a decision.

Possibilities for Realising Passive Metapolicies This could happen in various ways and three possibilities are presented and discussed here.

1. The enforcement of passive policies could theoretically be simulated with the help of active policies, as shown by the following description:

   (a) Each operation of the management system is guarded by an active policy. So it is only possible to execute an operation if the event which guards this operation is received. After execution of the operation, all events which could trigger these kinds of active policies are deleted in the system and an event is sent which notifies the terminated operation and the system state.

(b) Each passive policy is translated into an active policy which sends the events of all valid operations that can take place. The state of the system is encoded into one event which triggers this policy. This means that potentially 2^n events must exist if the system state can be described by n boolean state variables. This kind of policy is enforced before the one which guards the operations to ensure that all valid operations are permitted and executable.

(c) Only one operation is allowed to be executed at a time and the operation of the system is entirely described by passive policies.

It is immediately clear that this is not an implementable solution for supporting passive policies, because the system must be heavily altered and causes a large number of events. Also the requirement that the entire operation of the system is described by passive policies is not practical.

On the other hand, it shows in principle how passive policies could be realised. Whenever the management system wants to execute an operation, it must check whether there is a passive policy which would restrict the operation. This helps to find a better realisation.

2. In the item above, we have seen that before an operation takes place, it must be declared as valid through a passive policy. The system checks the metapolicies whenever it wants to execute an operation which could change the system state. The point when this must be done is called the *decision point* in the remainder.

   Practically speaking, it would be possible to add to each operation a check of its compliance with the stated passive metapolicies. This could be done by calling a special method of the management system or by sending a special event at each decision point. The decision point would be just before the actual execution when the operation knows its parameters. This is necessary, because both the operation and the method which checks the compliance must know the *context* of the actual execution. The context will include at least:

   operation This is the name of the operation which is executed if approval is given.

   affected objects The objects affected by this operation are listed to check whether there is a metapolicy which states something about them.

values of the attributes The actual values of the attributes of the objects which will be used by executing the operation.

The first two items of the context are needed to find all metapolicies which must potentially be applied and the last is used to check its conformance with the actual values used by the operation.

A good idea is to also specify the context of a metapolicy in its representation. This enables a more effective search for all metapolicies which must be considered.

The method which checks the compliance of the operation would give approval or deny the execution of the operation. In case of using an event mechanism, which includes the sending of the context, an event would be sent back to the method permitting or denying the execution.

The system and the operations, must be trusted in the way that each operation must implement the request for execution and if the execution of an operation is not permitted it will not be executed.

A variation of the principle described above is the introduction of a centralised method which controls and wraps each operation invocation, like a scheduler in an operating system. However, this scheduler does not consider time slices but the passive metapolicies. As a result, many of the already implemented methods need not be altered. This scheduler also needs the same information as described above, i. e. the context of the operation being executed.

Not considered here are solutions which use a variation of the two-phase commit protocol, which means that the operation is executed and afterwards the state is checked with the constraints of the metapolicies. If no objection is raised the changes are kept, otherwise they are undone.

3. A less flexible approach is to translate the passive metapolicies into static configuration files whenever possible. A very easy example is to generate an access control list for all passive metapolicies governing the access rights to the policy descriptions. This is only useful if an access control list is used for granting access.

   This approach cannot handle all sorts of passive metapolicies. An example is the chosen metapolicy from the paragraph before, except when can be ensured that some operations do not violate the invariant. In this case, a list of operations compliant with the metapolicy could be used for taking the execution decision.

It is clear, that the second and third items are two extremes. A real system could implement a mixture of these two, which would enable to consider time critical and flexibility aspects. Also using semi-translated/transformed

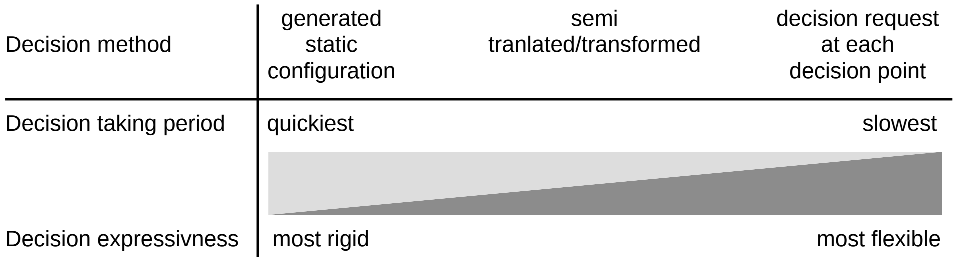

Figure 6.5: Spectrum of policy decision mechanims

passive metapolicies seems to be a possible solution, where not all details are static as figure 6.5 shows.

In general, all these enhancements to support passive metapolicies would also make it possible for the system to use passive policies with minor changes, because the basic concept of passive policies and passive metapolicies is the same.

Managed Objects

NoCScontrol is not a monolithic software application. It is realised by agent systems which are using CORBA objects. In subsection 6.1.3, the architecture of the system was described. The system consists of services which are realised by agents. The agents can act as clients using services from other agents, or as servers providing services for other agents. It is clear that the agents will be managed objects, because they provide the whole management functionality. They will be affected by metapolicies, both as targets and as subjects.

Furthermore, several other objects are created on an agent or are instantiated such as Enforcement Objects, Policy Objects, Persistence Service, etc. An advantage of the chosen implementation is that each class of objects and even every object itself can be referenced in a common and standard way. Not only the agents, but also the objects created on the agents can act in the client as well as in the server role.

For this reason, NoCScontrol fulfils the requirements for using metapolicies regarding the availability of managed objects without any enhancements.

State of Managed Objects

The state of objects must be observed to gather the necessary (management) information metapolicies use. Therefore, it must be clear which information is needed, how this information is collected, and how it is saved for later use.

The information needed for stating metapolicies is hard to predict. As mentioned in subsection 6.2.1, the expressiveness of a metapolicy depends on the available information in a management system. As a result of this, a certain amount of state information must be available. As in all management areas, the more and the detailed information a management system can use, the better a system can be managed.

The refinement metapolicy "A modified policy must be reviewed, before being activated again." entails the ability of observing the modification of a policy. If a policy is not immediately reviewed after the observation of its modification, the status, that this policy must be reviewed, must also be saved. A very flexible approach which allows the introduction of such information in addition to the metapolicy would be helpful. With introduction of the information, a mechanism for gathering the information must be specified.

The question arises which component is responsible for the observation and where is the collected information stored? Three solutions are possible in the NoCScontrol system:

objects are responsible In this model, the components are responsible for collecting and saving the data needed to preserve the state.

Each object saves its state in new attributes, which are accessible through an API. Every method which changes the state must also change the corresponding attributes.

This causes alteration of every component and every method in the NoCScontrol system and scales very badly. On the other hand, this adds the possibility of making every internal change externally visible.

agents are responsible In the architecture of NoCScontrol, the objects live on agents. Therefore, the possibility exists to make the agents responsible for the objects, which live on them.

The agent monitors all of its objects and processes, saves the state changes and makes them accessible through a standard API. Information which the objects cannot save by themselves must be saved by the agent.

adding a metapolicy support agent The tasks can be separated out to a new *Metapolicy Service Mobile Agent* responsible for monitoring and saving of states in the NoCScontrol system.

An approach where all components and objects send notifications of changes with the help of events has the greatest flexibility and scalability. The new metapolicy support agent would save the states whenever the metapolicies are interested.

Another advantage could be the integration of other metapolicy concerns, like the component needed for passive metapolicy support discussed in the second item on page 68. Also the management of the policy life cycle is possible, because this new agent has a macroscopic view on the management system such as its policy and enforcement objects and supporting processes.

In the remainder of this thesis I will assume that there is a Metapolicy Service Mobile Agent present.

Management Functionality

The management system is affected by management operations. This is especially the case with the enforcement of active metapolicies. They react on perturbations of the permissible state space. As a consequence, the managed objects and the management system must provide functionality which enables the enforcement of sufficient metapolicies.

In addition to the passive metapolicy support, functionality is needed for managing the components of NoCScontrol, the policy life cycle, and to control the supporting processes the management system uses. Currently, NoCScontrol only has a PDL parser as an supporting process. For this reason, there is hardly any management necessary. This will change, as soon as components for supporting the refinement, managing conflicts, etc. are added.

Metapolicy Service Mobile Agent must be able to correlate events, because it receives all the events sent by any method in the management system. Another possibility is to introduce a special event correlating component which lies between the event channel and the Metapolicy Service Mobile Agent. The problems resulting from event correlation will not be considered here.

The agent provides methods which allow changing the state of the management system. This can serve, in case of active policies, as a middleware between the Enforcement Objects and the methods provided by each component. This allows metapolicies and thus the Enforcement Objects to use special management actions whenever actions more complex than the existing primitive methods are necessary or desired. These specialised management actions are macros, accumulation of primitive method calls, or newly implemented methods. In the refinement of active (meta-)policies these management actions can be developed as the implementables from [Goh 97] described in subsection 2.4.1 and 2.4.3.

The NoCScontrol system with its flexible architecture can include various kinds of supporting objects and processes. The Metapolicy Service Mobile Agent provides support for managing these new objects or processes whenever necessary.

On the other hand the agent must keep track of all the states it is responsible for, which was described in the section before. Therefore, it must have implemented the functionality for this purpose, and for sending notifications which the Enforcement Objects are using for triggering active policies. Querying the state of NoCScontrol by an external component like a graphical user interface should be possible. This entails that the agent acts as a server and provides an API for for this task.

An interface which supports adding management operations would also be helpful, especially during the refinement process, but this addition is not discussed here.

The Metapolicy Service Mobile Agent acts as a central component responsible for the management of the NoCScontrol system.

Policy Representation

Here, the enhancements necessary for the adequate representation of metapolicies are described. In addition to this, it is discussed which part of the NoCScontrol system must be altered to use the enhanced representation possibilities. An exception is the Policy Interpreter, which will be discussed in the metapolicy enforcement.

Subjects The semantics of the subjects in the policy specification with the PDL is somewhat in between the semantics used in authorisation policies and obligation policies. Authorisation policies state whether a subject is allowed to do something and in the case of obligation policies the subject is obliged to do something.

In the NoCScontrol system, the subject is used to check whether a policy must be enforced. The subject specifies the home domain of a nomadic system. When an event is caused by a nomadic system its home domain is compared to the specified subject of each policy. Matching of both the subject and the home domain causes the enforcement of the policy. An event is for instance the request to use a special resource, as printers, email, etc. This means that actions are taken for granting access of the nomadic system. The subject is at least additionally[5] used for stating an authorisation policy, but in the same policy actions are taken for granting access.

As a consequence of the nomadic computer system scenario, there is no concept of roles. The roles are necessary to state for instance the responsibility of actions.

Another result is the implicit restriction that the subject specifies the home domain of a nomadic computer system is not valid any longer.

Targets At the moment, all management actions specified in the action part of a policy are executed on the specified targets.

[5]The constraint expressions of the policy are evaluated before the policy finally is enforced.

Only one type of targets can be specified through the CORBA Naming and Topology Service. If the subject is a role and thus responsible for actions, these actions are not necessarily invoked on only one type of targets. Each management operation can be invoked on different managed objects. This is necessary for supporting the creation of management operations on the Metapolicy Service Mobile Agent as mentioned before.

Events The use of an additional *Event Definition Language (EDL)* [Koch 97] adds flexibility to the system. Unfortunately, the Policy Service must be enhanced to support the EDL. The EDL allows the specification of events separately from the policy specification. In conjunction with the Metapolicy Service Mobile Agent, the EDL can also be used to specify the events the Metapolicy Service is sending.

Constraints As a consequence of changing the semantics of the subject, checking of the home domain is done with an additional constraint expression, which regulates the enforcement of a policy.

The discussion of the invariant properties of passive policies has shown the importance of adequately specifying the policy. For this purpose predicates must be available as can be seen in the invariant example in table 6.3. When adding new state variables to the Metapolicy Service Mobile Agent, adequate primitives for implementing the necessary predicates must be considered.

The Metapolicy Service Mobile Agent implements a large number of additional predicates, because it holds the states of components and objects. These states are notified by events and saved by the Metapolicy Service. Therefore, it is a central component which can provide these predicates and constraints. The advantage is that not every metapolicy must re-implement complex constraints again.

Let´s consider the example when two actions are in execution and a metapolicy wants to prevent the execution of a third one. The beginning of the execution was notified by the actions and recognised by the Metapolicy Service. The Metapolicy Service also provides a predicate `inconcurrence(Action A, Action B)`. This can be used in the specification of the constraint part of the metapolicy to decide whether the third action can be executed.

The Constraint Interpreter by the Enforcement Object Factory Mobile Agent (figure 6.4) must be able to process and evaluate the stated constraints. Therefore, it must be enhanced to allow the constraint evaluation in cooperation with predicates of the Metapolicy Service. The Constraint Interpreter makes use of a decision service provided by the metapolicy agent. This service evaluates the potentially complex predicates for the Constraint Interpreter. This prevents the repeated implementation of the predicates and allows their central management. In a later step, this service could probably be separated out into a stand-alone component. This would allow normal policies to use this service for their purposes.

Actions The Action Interpreter (figure 6.4) is enhanced to allow the use of methods of every CORBA server as an action. Multiple actions from several services can be used. The realisation of conditional actions adds convenient flexibility. Future enhancements can add concurrence and job control.

Metapolicy Enforcement

The Policy Interpreter is enhanced in a way that it distinguishes between normal policies and metapolicies. This is necessary, because metapolicies have the power to control almost every part of the management system and therefore they must be recognised. This is the first step to integrate security concerns into the system. Despite of this, the Policy Interpreter binds the metapolicy objects to the appropriate subjects and targets. Every one of them can be the Metapolicy Service Mobile Agent, i. e. the metapolicy objects must be bound to it. The Policy Interpreter activates operational policies and creates the necessary Enforcement Objects with the help of the Enforcement Object Factory Mobile Agent.

The changes necessary for the general support of passive metapolicies have already been discussed from page 67 onward. Here, only the Policy Interpreter is discussed to accomplish the passive metapolicy support.

In case of passive metapolicies, the Policy Interpreter does not create an Enforcement Object but integrates the metapolicy into the Metapolicy Service Mobile Agent. The necessary steps are first to register the events of interest sent at a decision point from Mobile Agents, then to specify the answering events from the Metapolicy Agent, and finally to integrate the description how to calculate an appropriate decision and determine which events must be sent.

The following section will give a model for the core of the suggested enhancements.

6.3 Implementing Metapolicies in NoCScontrol

This chapter gives a detailed description of a basic implementation of metapolicies in the NoCScontrol management system. Not all possibilities described in section 6.2 are implemented, because this would entail a large number of modifications. The implementation shows that basic metapolicy support is possible within the realms of the management system and with reasonable changes.

6.3.1 Supporting Processes

The NoCScontrol system has limited support by supporting processes since since it is a small prototype for policy-based management of a nomadic computer system scenario. This can be seen from the policy life cycle in figure 6.1. There is no real distribution process, no test process, and no delegation process, because responsibilities are not considered and used by the system.

Refinement is only supported by providing three different kinds of policy objects. These follow the three levels of policies as described in [Koch 97]. There is no refinement process in terms of a description or computer support. The focus of the system is the enforcement and for this purpose an automated enforcement of obligation policies[6] is implemented.

For this reason, the implementation of metapolicies provides control in the areas of refinement and enforcement. This allows the specification of metapolicies in the categories refinement, conflict, exception, and enforcement.

6.3.2 Architectural Changes

A *Metapolicy Service Mobile Agent* is added to the already existing agents, as shown in figure 6.6. Like the existing agents, it uses the CORBA environment, e. g. for accessing services from other CORBA components, and for event communication. It also uses the Constraint Interpreter, but this is not depicted in the figure.

The *Metapolicy Supporting Service* is used to store the states which are not saved by the objects themselves. This is accomplished by active metapolicies which invoke methods of the Metapolicy Supporting Service for storing and deleting states variables. The states are stored in the *Object State Database*.

In addition to this, the context of actions before their execution is stored by the Metapolicy Supporting Service as an *action event* in the *Action Event Database*. The changes made enable the system to control the enforcement process of policies.

Passive policies are not yet supported by the NoCScontrol system. For this reason, they are directly realised by the Metapolicy Service Mobile Agent. It provides functionality to store, retrieve, add, and save them. They are stored in the *Metapolicy Database*. Decisions governed by passive meta-

[6]In subsection 6.2.3 an ambiguity of the semantics between authorisation policies and obligation policies as used in the NoCScontrol is discussed.

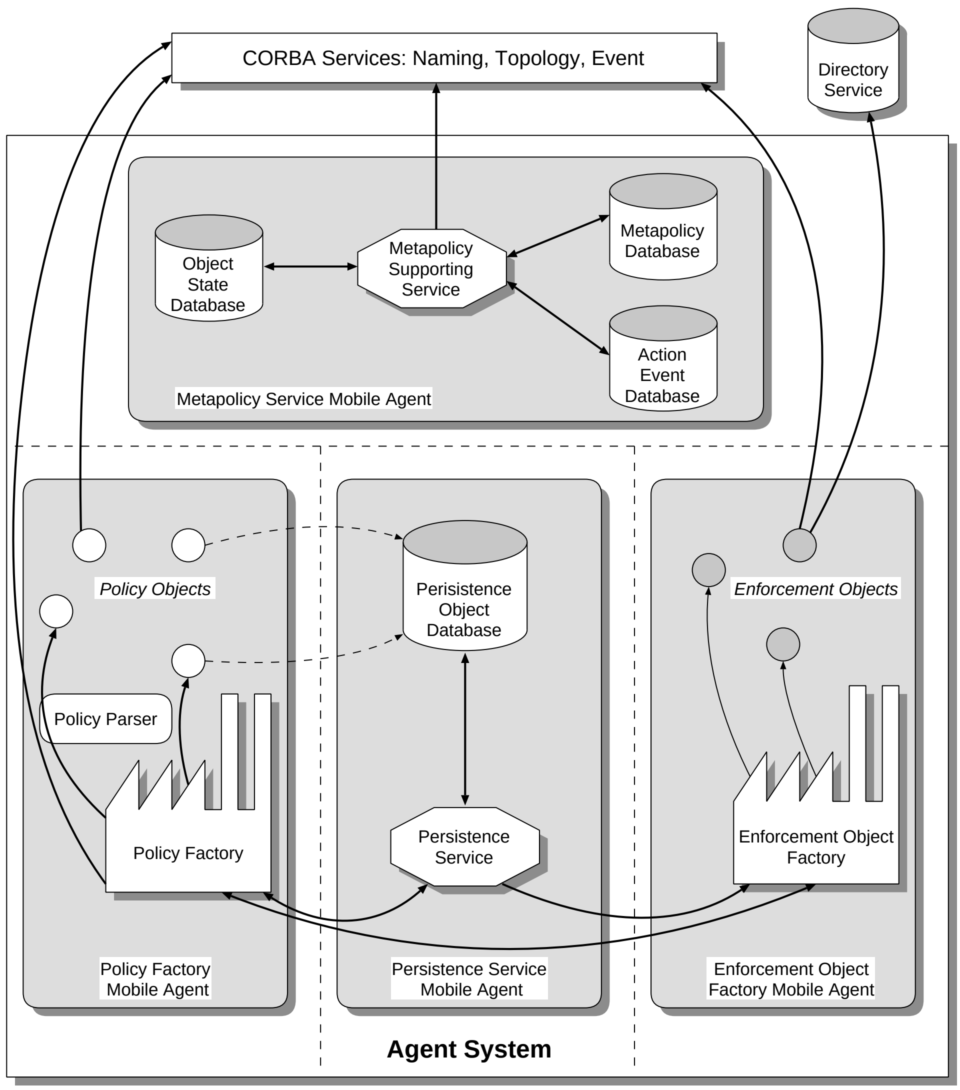

Figure 6.6: Architectural overview of the metapolicy enhanced management system

`policy_enforcement:sourcePolicyID=PolicyX (!(targetAgentID= EventDB:policy_enforcement:sourcePolicyID=PolicyY:targetID))`
`policy_enforcement:sourcePolicyID=PolicyY (!(targetAgentID= EventDB:policy_enforcement:sourcePolicyID=PolicyX:targetID))`
`policy_enforcement:sourcePolicyID=* (!(nomadicSystemDomain="NT"))`
`policy_enforcement:sourcePolicyID=* (!(ObjectStateDB:*="to_review"))`

Table 6.4: Passive metapolicy examples enforced by the Metapolicy Service Mobile Agent

policies are taken by the Metapolicy Supporting Service with the help of the Constraint Interpreter of the NoCScontrol system.

6.3.3 Metapolicy Examples

In this subsection, examples of metapolicies and their structure are presented. Their enforcement guides the presentation of the enhancements of the NoCScontrol system.

Table 6.4 shows three passive metapolicies. The first one is taken from table 5.4, the second and third are taken from table 5.5. They were explained in detail in subsection 5.5.4 and in subsection 5.5.5.

The policies have the following structure:

- Each constraint for a policy appears on a separate line.

- Each line begins with a textual event identifier in the form described in [Radi 98]. The event name follows a ‘`:`’ and one body entry name of the event and a ‘`=`’. After that, a real value or a ‘`*`’ must be given. Next, a space follows as a separator. This specifies the *triggering event* used here to decide whether the passive metapolicy must be considered.

- The triggering event is followed by a valid constraint expression which can be processed by the Constraint Interpreter. In the constraint, a state of an object in the Object State Database is compared to a given value. Or, in the case of the Action Event Database values of action events can be used. Values in the constraint are referenced by the database name followed by a ‘`:`’ and a *reference key*.

  In case of the Object State Database the ‘`:`’ is followed by an object reference or a ‘`*`’ as the reference key. In case of the Event Database, a triggering event, a ‘`:`’ and one event body entry must be given. This references the first value. Then a ‘`=`’ is specified followed by a real value or a second reference of a value. In case the reference key is omitted and only an event body is specified, the event body of the triggering event is used. The asterisk matches any value.

```
POLICY MetaPolicy_SetReview_in_alteredPol
FOR MetapolicyServiceMobileAgent
ON PolicyFactoryMobileAgent
DELIVERS execution{string action_name; string PolicyID}
ACTION
 {Subject.insert_object_state(execution:PolicyID, "modified");
  Subject.insert_object_state(execution:PolicyID, "to_review"}
CONSTRAINT {execution:action_name = "setOperPolDescription"}
```

Table 6.5: Active metapolicy example in PDL

- All constraints for one triggering event are combined together by a logical `AND`. The metapolicy grants an action when all combined constraints evaluate to `true`. Therefore, the constraints must be stated in an appropriate form.

The first line of table 6.4 reads as follows: Add the constraint that "`PolicyX` and `PolicyY` are not together enforced on the same target as" to the constraint of granting `PolicyX`. The line next to the last is similar and may be read as: Add the constraint "no enforcement of policies which are for nomadic systems in the domain `NT`" to all policies. This shows that the '`*`' matches all policies. The last one specifies: "No enforcement of policies with are in the state `to_review`."

The first two lines are the refinement of one policy and both lines are necessary, because during the enforcement of a policy the metapolicy is retrieved by its policy name. The metapolicy is symmetric, therefore only `PolicyX` is exchanged by `PolicyY`.

The metapolicy in figure 6.5 is an active metapolicy. It is conforming to the PDL of the NoCScontrol system. As opposed to a normal policy, the subject is the Metapolicy Service Mobile Agent but could be any other management agent. It need not be a nomadic system. No new keyword such as `METAPOLICY` instead of `POLICY` was introduced to allow for reusing of the complete Policy Factory Mobile Agent without any modifications concerning the policy processing. Instead, the policy name is prefixed by `MetaPolicy_`.

The policy could be one of the policies resulting from refinement of the third metapolicy in figure 5.1. It says: When the action `setOperPolDescription` is executed by the PolicyFactoryMobileAgent on a policy, memorise the state of this policy as `modified` and as `to_review`.

In the next subsection, it is shown how these metapolicies are generally enforced.

6.3.4 Policy and Metapolicy Enforcement

The altered enforcement process resulting from the changes of the NoCScontrol is described in this subsection illustrated by the metapolicy examples.

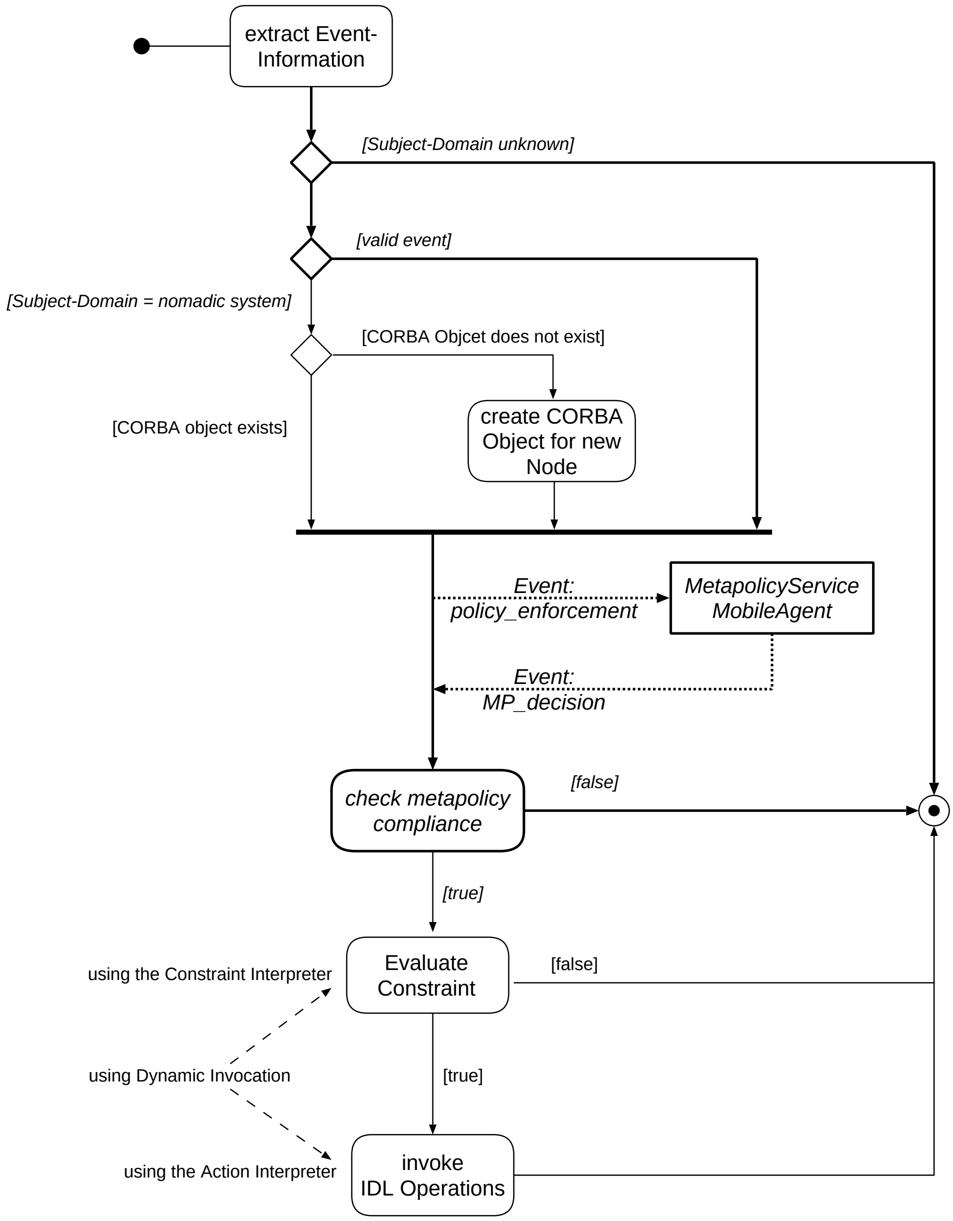

Figure 6.7: `Enforcement Object` dynamic model diagram

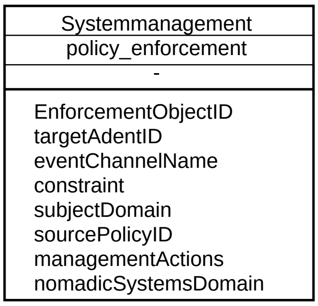

Figure 6.8: Structured `policy_enforcement` event

Figure 6.7 shows the changed dynamic model of the Enforcement Objects of normal policies and metapolicies. Additional state transitions, events, and decision points are made visible by stronger lines and *slanted typeface*.

In comparison to the former enforcement process, subjects other than nomadic computer system are possible, and before a policy gets finally enforced the Metapolicy Service Mobile Agent is asked for approval. For this purpose, each Enforcement Object sends an `policy_enforcement` event as shown in figure 6.8. The `policy_enforcement` includes information specifying the state of the Enforcement Object. This provides the context in which a policy is enforced by an Enforcement Object and on this basis, it is possible for the Metapolicy Service Mobile Agent to take a decision.

The Metapolicy Service Mobile Agent sends an `MP_decision` event back to the Enforcement Object. This contains the decision for the enforcement of the policy. At this phase of the implementation, it only grants enforcement by `true` or denies it by `false`. Enhancing both, the capabilities of the decision process and the Enforcement Objects, will allow further events. The Enforcement Object executes the actions specified by the policy when it receives a `MP_decision` containing `true`. Otherwise, the enforcement stops.

The enhancements, which allow subjects others than a nomadic system, are necessary for the enforcement of active metapolicies. The process of granting enforcement by the Metapolicy Service Mobile Agent is needed for the realisation of passive metapolicies.

The remainder of this subsection describes the application of the example metapolicies from subsection 6.3.3.

The active metapolicy is stored in an Operational Policy object by the Policy Service and enforced in a very similar way as normal policies by NoCScontrol system. The only differences is the different subject, namely the `MetapolicyServiceMobileAgent`, which performs actions not for managing nomadic computer systems, but for managing the management system. It invokes the `insert_object_state` method, which creates an entry for the policy object, if necessary, and first stores `modified` and, with the second invocation, `to_review` in the Object State Database. The Policy Factory

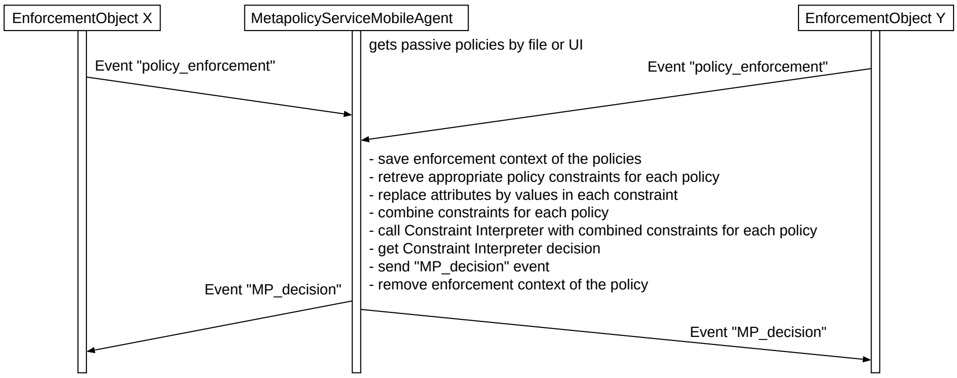

Figure 6.9: Enforcement of policy or metapolicy

Mobile Agent is modified to send an `execution` for each operation invoked on them. The constraint checks which method was called and triggers the policy actions when it was the `setOperPolDescription` method.

The passive metapolicies are enforced by the Metapolicy Service Mobile Agent. Figure 6.9 shows the enforcement of two normal policies and passive metapolicies. The two normal policies starting the enforcement process nearly at the same time. This will result in concurrent enforcement of both policies. The two Enforcement Objects send their state by an `policy_enforcement` event. This is saved in the Action Event Database. After that, the Metapolicy Database is searched for metapolicies governing the policies. The constraints of the metapolicies found are combined by logical `AND` and one large structured event is created. This is necessary, due to limitations of the Constraint Interpreter, which can only be called with one structured event. The result evaluated by the Constraint Interpreter is sent by an `MP_decision` event to the Enforcement Object of the normal policy.

After the `MP_decision` event has been sent, the `policy_enforcement` event is removed from the Action Event Database. This implies that an action granted by an `MP_decision` event is executed immediately which is clearly not true. This approach was taken to avoid the use of several events, notifying the beginning and ending of an action, and to avoid large changes in the existing system. This may restrict the use of metapolicies to govern concurrent events, but these are not considered by the NoCScontrol system either at the moment. Using events in general for notifying a metapolicy decision makes the decision dependent on the semantics[7] of the event notification mechanism. This problem is also not considered here.

The following subsection describes the developed design in greater detail.

[7] i. e. best effort, exactly once, at most once, etc.

6.3.5 Implementation Details

General considerations, the enhancements to existing classes, and the class diagram of the Metapolicy Service Mobile Agent are presented in this part.

Additional attributes and classes are depicted by *slanted typeface* in the class diagrams. An exception is the class diagram for the Mobile Service Mobile Agent, because this has completely been developed.

There is no real refinement process and thus no appropriate decision points in the NoCScontrol system. For this reason, the passive metapolicy enforcement is not implemented by the Policy Factory Mobile Agent. It can be realised in the same way as it is done in the enforcement process.

The Object State Database and the Action Event Database conceptually store the state of objects. Two different databases are used to support an efficient implementation. For the realisation of new state attributes, the mechanism of policies already implemented by NoCScontrol is used. This allows maximum flexibility in choosing the number and type of attributes for the objects. The policies need a public API to call methods by CORBA. Therefore, the entries of the Object State Database are added and removed with the help of such public methods. Also, the entries in the Object State Database are more “stable” in contrast to the entries in the Action Event Database. The entries of the Action Event Database are received by events. These events are stored directly in Action Event Database.

Decision for the State Observation Mechanism

The Metapolicy Service Mobile Agent must be made aware that an action is executed which can change the state of objects. The decision for the approach implemented is founded by the following considerations using the alteration of a policy object as an example. Various possibilities exist to realise the observation:

- An agent monitors the policies and keeps track of changes of the policy description. If an alteration is detected, it produces a notification.

  An advantage of this is that no changes to the system are necessary such as introducing a `modified` flag or changing methods like in the following suggestion, because an external agent could monitor this independently. A disadvantage is the need for an additional component in the system.

- The program or method which is used to alter the policy description changes the state of the policy to `modified`. An agent checks this state and produces a notification.

  This differs from the one before in that the state of modification of a policy is somewhere explicitly noted, e. g. in the agent, or by an (extra) attribute of the policy.

The advantage over the first method is the fact that the state lives longer than the event. Components which are not able to receive the event due to a failure of transmission or just due to the fact that the components are not started can check the state of the policy on their own. A disadvantage of this method is clearly the fact that the state of the policy must be stored somewhere. Using this mechanism could result in changes of the agent or policy, which makes it very expensive to add other state variables. Reduced flexibility may result from this.

- The program or method which is used to alter the policy description produces the notification by itself. This is done by calling a special method with the name of the method and its parameters.

  Here, no additional components are necessary, but each program or method which changes a part of the policy is responsible to produce a notification. If changes are only possible through one API, which would be not a massive restriction, only the methods implementing this API must be adjusted. In case of the NoCScontrol the parameters are almost always typed as strings making the invocation of a special method easy. Due to the small numbers of public methods, this approach is feasible. The integration of a proxy mechanism for a large existing system may be a better solution in this case.

The third possibility was chosen due to its advantages and the disadvantages of the others. The notifications are realised with the help of events.

Enhancements of the PolicyFactoryMobileAgent class

In figure 6.10 the additions to the `PolicyFactoryMobileAgent` class diagram are depicted.

Attributes The new attributes `_MP_PushConsumer`, `_MP_ProxyPushSupplier`, `_Pol_ProxyPushConsumer`, and `_Pol_PushSupplier` are used to hold the necessary objects for the event communication. The channels are created by the Metapolicy Service Mobile Agent.

Methods

- A public method `PolicyFactoryOperation` is added to the interface specification `setOperPolDescription`. This was done due to limited functionality which only allows creating and deleting policies. This method sets a new policy description in PDL within an existing policy object. Both the policy object identifier and the new description, are given to the method as a parameter. Successful execution returns `true` otherwise `false`. In case of successful execution, the new method calls

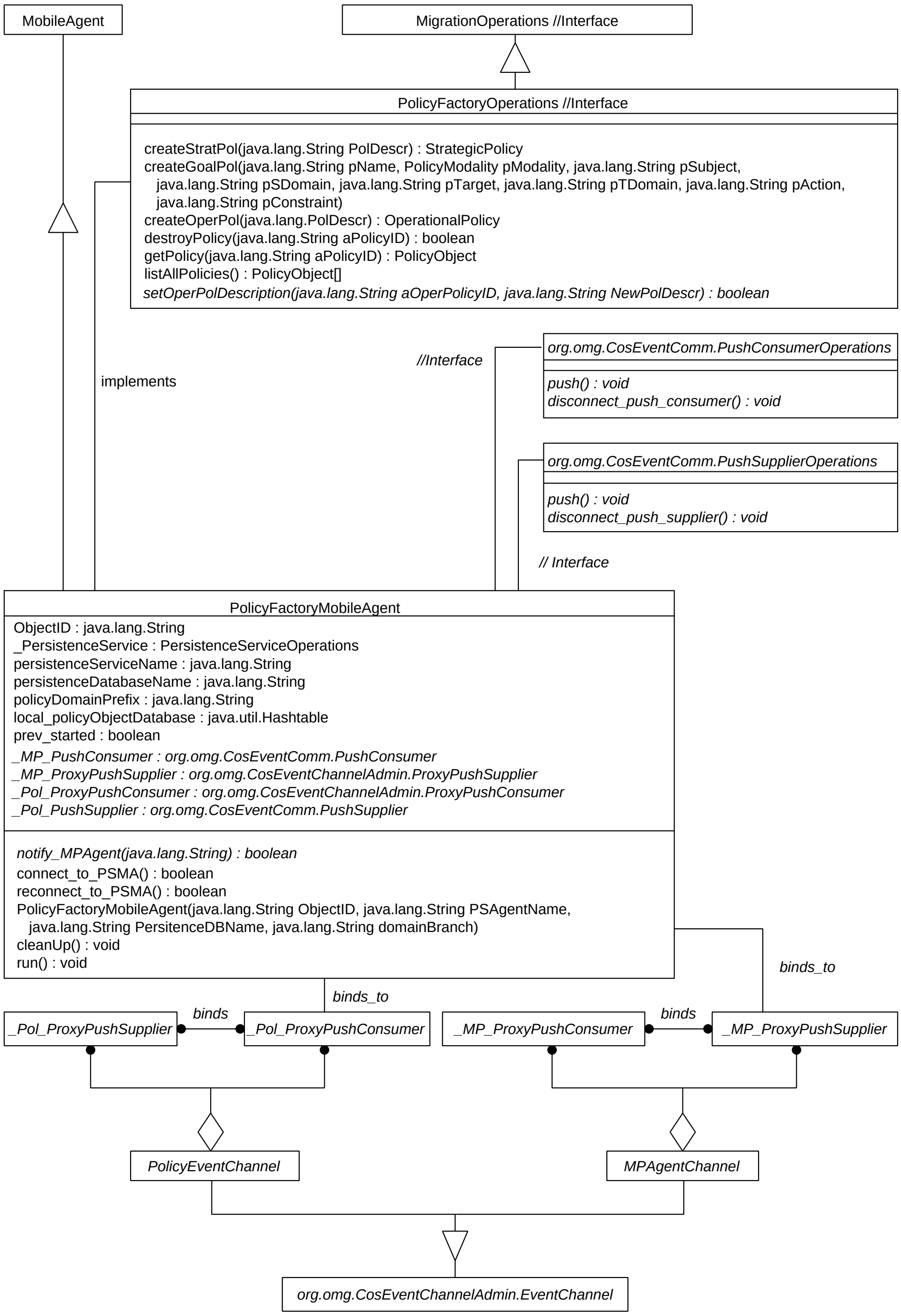

Figure 6.10: Enhanced `PolicyFactoryMobileAgent` class diagram of the NoCScontrol system

the notify_MPAgent method to notify its execution before it returns. It is also used by the active metapolicy example. With this new method, the notification mechanism incorporated by the notify_MPAgent can be tested without touching existing methods.

- The private method notify_MPAgent gets the name of a method and its parameters typed as strings and creates an execution event. This event is a structured event as described in [Radi 98]. It sends the event to the Metapolicy Service Mobile Agent.

  The use of parameters typed as strings is not a restriction at the moment and is used for convenience. With changes to the system in the future, a more generic mechanism must be incorporated allowing a generic notification of (internal) state changes. Without any changes of each method, the *tie mechanism* [Vis 97a, Radi 98] could be used for sending the event. This possibility is not used here because the tie mechanism is only used to simulate multiple inheritance in Java. Therefore, it is supposed to be transparent. Adding functionality to the created tie classes violates this assumption.

- The push method is implemented for the use of the two event channels, PolicyEventChannel and MPAgentChannel. The use of the two event channels is terminated by the method disconnect_push_supplier and disconnect_push_customer.

Enhancements of the EnforcementObjectImpl class

In figure 6.11 the modifications of the EnforcementObjectImpl class diagram are shown.

Attributes The new attributes _MP_PushConsumer, _MP_ProxyPushSupplier, _EO_ProxyPushConsumer, and _EO_PushSupplier are used to hold the necessary objects for the event communication. The channels are created by the Metapolicy Service Mobile Agent.

Methods The subject of an operational policy a nomadic system by default. In case of metapolicies, the subject is a component of the management system, for example the Metapolicy Service Mobile Agent. The subject domain is no longer only the home domain of the nomadic system, but can also be the domain of the management system or its agents. The method push is changed to achieve this. Also, the different event channels are handled in this method.

In addition to this, it sends, receives, and checks the Policy_enforcement event as shown in figure 6.7.

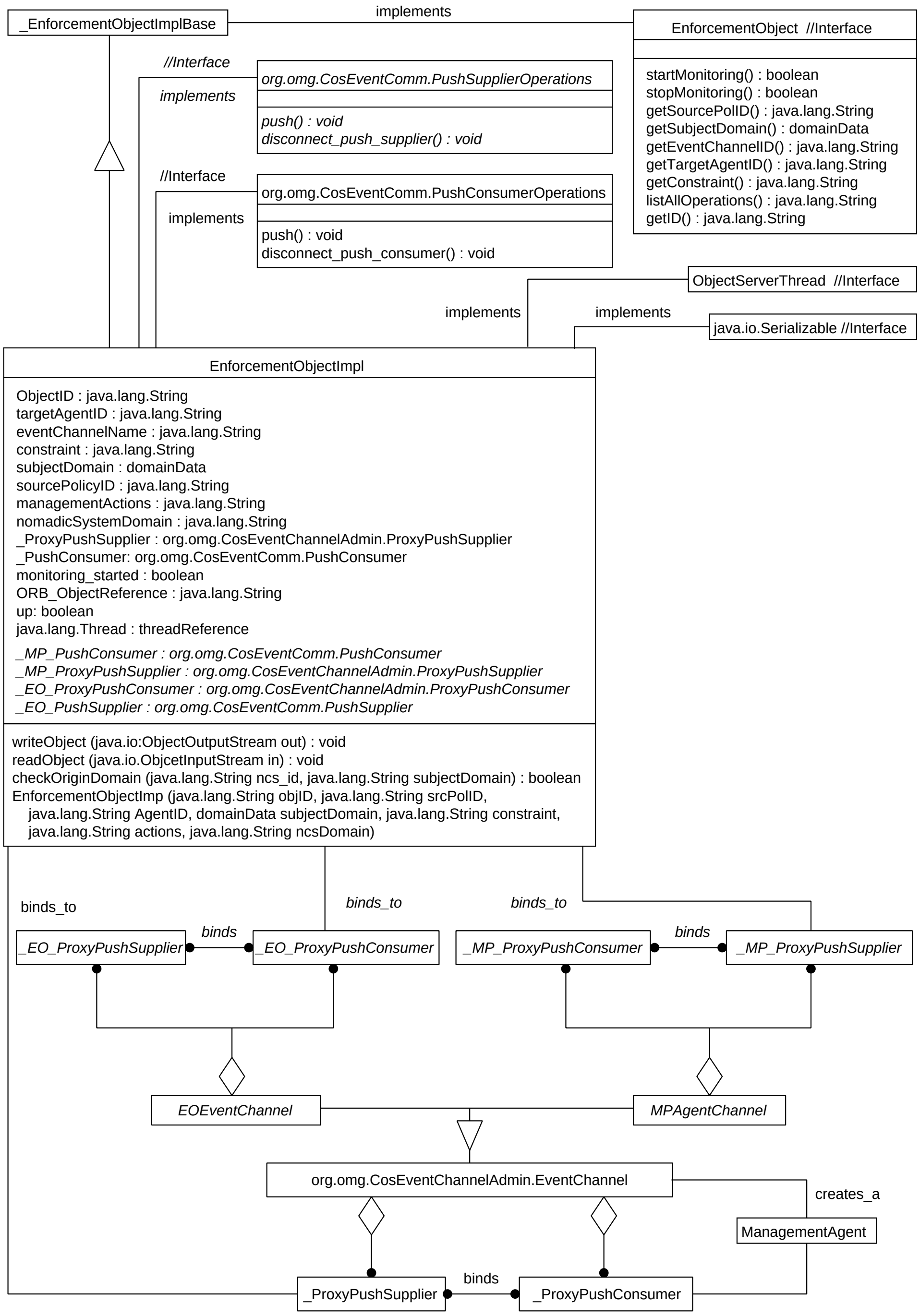

Figure 6.11: Enhanced `EnforcementObjectImpl` class diagram of the NoCScontrol system

All this functionality was added to the `push` method because it was done so in the NoCScontrol system. The method was already very large and complex. A good idea would be to separate the functionality other than for communication out of it.

The Metapolicy Service Mobile Agent

The class diagram of the Metapolicy Service Mobile Agent is shown in figure 6.12. It creates the additional event channels and enforces the passive metapolicies.

For this purpose, it provides the functionality to integrate passive policies in the Metapolicy Database. The enforcement of the passive metapolicies is started with receiving a triggering event. The appropriate constraints are retrieved and then combined with logical `AND`. This constraint is encapsulated in a structured event, because the Constraint Interpreter expects one as parameter. The Constraint Interpreter is called for evaluating the constraint. Its result is sent back to the requesting object by a structured `MP_decision` event. This event contains either `true` or `false`. In case of no state information available, `true` is sent by default for not blocking the enforcement. The result of the Constraint Interpreter is sent by an event to the requesting object.

A user interface, which is not described here, is used to change passive metapolicies, or to change and query the states of objects.

The entries in the Object State Database are inserted and removed by policies or the user interface. The events in the Action Event Database are inserted by reception and deleted after sending the answering `MP_decision` event. This has some unwanted drawbacks which are described on page 82.

As in the NoCScontrol, the structure of all events the agent receives or sends must be known.

Separate event channels are created by the Metapolicy Service Mobile Agent allowing other components to register with channels of their interest. Considering security concerns in the future make it possible that security relevant data is passed through the MPAgentChannel other data through the agent specific channels.

Attributes

- The `ObjectID` is the compound name of the agent and has the form described in [Radi 98].
- In the `object_stateDatabase` the states for objects are stored. An object can have several *different* state attributes, such as `modified` and `to_review` from the example metapolicy.
- The `action_eventDatabase` holds the structured events known by the metapolicy service.

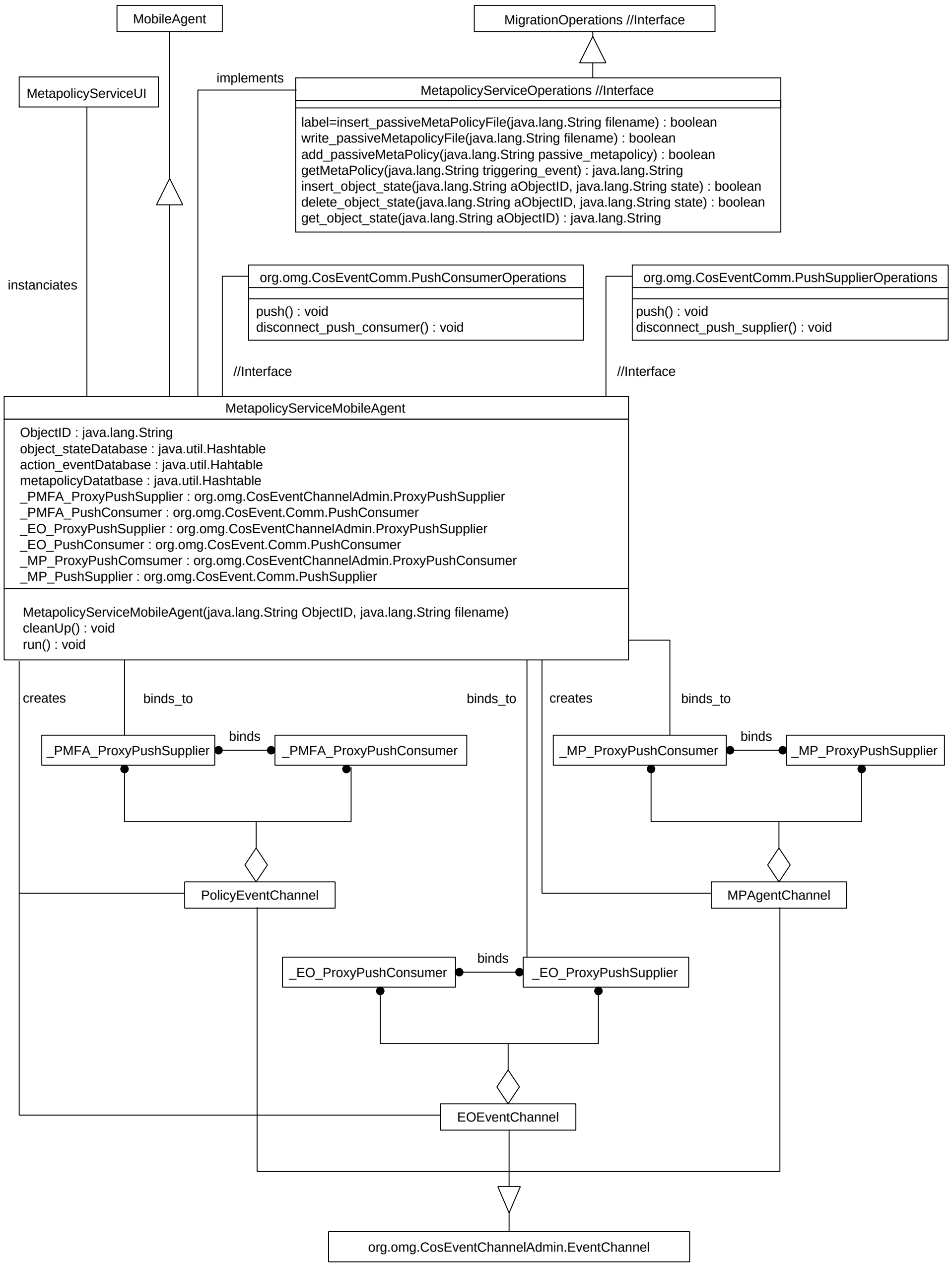

Figure 6.12: `MetapolicyServiceMobileAgent` class diagram

- The `metapolicyDatabase` holds the passive metapolicies.
- The remaining attributes are necessary for creating the event channels by the Metapolicy Service Mobile Agent.

Methods

- `insert_passiveMetaPolicy` and `write_passiveMetaPolicy` load passive metapolicies form a file into the Metapolicy Database or write them back. `add_passiveMetaPolicy` adds a single metapolicy. `getMetaPolicy` retrieves a passive metapolicy for a triggering event as described in subsection 6.3.3.
- `insert_object_state` inserts its state in the Object State Database for a given object. This method must be idempotent, i. e. repeated inserting of the same state of an object must not change the object's state entry. `delete_object_state` removes a state entry of an object and removes the object from the database when it is the last entry. The `get_object_state` method retrieves the state of an object from the database.
- `MetapolicyServiceMobileAgent` is the constructor method of the agent and is initialised with an object identifier and a path to a file containing valid metapolicies.
- The methods `run` and `cleanUp` must be implemented to follow the structure and constraints described in [Radi 98].

6.3.6 Scalability and Possible Enhancements of the Implementation

The implementation of metapolicies in the NoCScontrol system is done by reusing as much as possible of its architecture and infrastructure. Adding of new IT management functionality with additional agents does not negatively affect the use of the implemented metapolicy capabilities. For this reason the system can scale in this dimension without any changes.

Negative performance issues as a result of the additional event communication and decision process may be seen as the main drawback of the flexible implementation approach used here. Other possible approaches for performance critical areas may be used in this case, e. g. translation into configuration files.

Some changes are done to provide the necessary information for the use of metapolicies, others to allow metapolicies take decisions. These changes are developed in a way which allows their integration in any further enhancements of the NoCScontrol application. Especially newly developed processes can be managed with metapolicies. During the development of new components, management functionality for these components can be integrated. Metapolicies will make use of this management functionality.

The Metapolicy Service Mobile Agent integrates the facilities for memorising additional states of objects, mainly to realise passive metapolicies. Metapolicies will use this functionality only for their management purpose. It is also possible to use this functionality, e. g. the created event channels and the Object State Database, for the sake of normal policies and management. This may entail security concerns, but these are not considered in this thesis.

The use of metapolicies in a management system must be restricted. Marking a metapolicy in the representation explicit with `METAPOLICY` instead of `POLICY` is a first useful step to handle different security concerns. The fact that security concerns are necessary is evident. Metapolicies can not change the behaviour of the management system, and access to all existing policies is possible and necessary. To support a sophisticated security model much more changes are necessary, but this is out of scope for this thesis.

The external representation of passive metapolicies in plain ASCII files allows to structure them in the classes as presented in section 5.4. It is also possible to generate the metapolicy containing files with external tools. Tools may also be used for other purposes, e. g. detecting metapolicy conflicts or simplifying the constraint statements.

The functionality to create new predicates or constraints as mentioned on page 74 is not implemented. A combined design process for enhancing the Constraint Interpreter and the Metapolicy Service Mobile Agent is necessary to achieve best integration.

In further enhancements, the use of event correlation may be desirable to reduce the number of events the metapolicy service has to process. For this reason, explicit marking of metapolicy events may be necessary to distinguish them from policy events.

The ability to generate events in a metapolicy would make it possible to trigger other policies or metapolicies (chaining). The specification of sequential and concurrent enforcement of (meta-)policies would be straightforward to realise. Additionally, notifying state changes of objects by the Metapolicy Service Mobile Agent makes is possible to trigger other active policies and metapolicies.

6.4 Summary and Conclusion

This chapter showed the integration of the developed metapolicy concept with the help of an existing policy-based management system. It was one of the design principles not to change large parts of the existing system. This showed the possibility of the integration of metapolicies, although a consequence is, that not all of the possibilities of the metapolicy concept can be implemented.

Constraints in the existing management system restricting the possibilities of the metapolicy concept were discussed. Different approaches to enhance the system further than the given implementation with their advantages and disadvantages were presented. They can be considered in future enhancements of the NoCScontrol system.

Chapter 7

Summary and Outlook

The concept of a *metapolicy* has existed for quite a while, but in the literature, its detailed significance has not been thoroughly explored, especially in relation to the management of information technology systems. I have investigated the processes related to the application of management policies. The understanding gained as a result of this investigation led to an enhanced policy life cycle. Using this knowledge, significant insights into the metapolicy concept were derived.

The introduction of the supporting processes and their relation to the states in the policy life cycle has an impact on the understanding of policy-based management as well. Most importantly, in addition to the common understanding that metapolicies are "policies about policies", I introduced the concept that metapolicies are also "policies governing the policy life cycle".

Although the classification of metapolicies has been proposed previously, I take a different approach by looking at management needs. In my classification, I am able to identify the mapping of the type of metapolicies to the supporting process associated with a state in the enhanced policy life cycle. In this way, this classification can become a basis on which a self-contained management system can be developed.

The requirements for metapolicies in policy-based management systems were examined:

- The management system must contain appropriate management information.
- It must be possible to address the components in the management system and the components, in turn, must provide methods that allow them to be managed. These methods should be accessible by a policy mechanism.
- In order for the metapolicies to control a metapolicy-compliant operation of the system, the metapolicies must be strictly followed by the system itself when it is running.

Considering these requirements a proposal for the integration of metapolicies in an existing policy-based management system was made. This proposal was transformed into a detailed model allowing the implementation within the existing system.

A starting point has been created for future work, which will include the design of an appropriate representation to describe metapolicies more completely, to implement the enhanced policy life cycle with its related processes, and to test metapolicies on those processes. Once this has been achieved, a useful extension of the presented metapolicy concept could be the inclusion of a mechanism to control the exchange of policies between different management systems.

Abbreviations

ACL	Access Control List
AIS	Automated Information System
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CORBA	Common Object Request Broker Architecture
DHCP	Dynamic Host Configuration Protocol
EDL	Event Description Language
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IDL	Interface Definition Language
ISO	International Standards Organisation
IT	Information Technology
MAC	Medium Access Control
MIB	Management Information Base
MO	Managed Object
MP	Metapolicy
NIS	Network Information Service
NoCS	Nomadic Computer System
OMG	Object Management Group
OMT	Object Modelling Technique
ORB	Object Request Broker
OSI	Open Systems Interconnection
PDL	Policy Description Language
QoS	Quality of Service
RFC	Request for Comment
RPC	Remote Procedure Call
SSO	System Security Officer
UI	User Interface
URL	Uniform Resource Locator
WWW	World Wide Web

Bibliography

[BaBa 98] Rosy Barruffi and Claudio Bartolini. Information Models for Management: An object-oriented logic-based Approach. In *Proceedings of the 5^{th} Workshop of the OpenView University Assosiation: OVUA'98*, Rennes, France, April 1998.

[BaMi 98] Rosy Barruffi and Michela Milano. Interactive Constraint Satisfaction Techniques for Information Gathering in Planning. In Henri Prade, editor, *13^{th} European Conference on Artificial Intelligence (ECAI-98)*, pages 514–515, Brighton, UK, August 23–28 1998. John Wiley & Sons, Ltd. Young Researcher Paper.

[BFA 97] Elisa Bertino, Elena Ferrari, and Vijayalakshmi Atluri. A Flexible Model Supporting the Specification and Enforcement of Role-based Authorizations in Workflow Management Systems. In *Proceedings of the Second ACM Workshop on Role-Based Access Control*, Fairfax, Virginia, USA, November 6–7, 1997.

[BLMM 97] Rosy Barruffi, E. Lamma, Michela Milano, and P. Mello. Application of Planning Techniques for System Configuration Tasks. In *Proceedings of the 4^{th} Workshop of the OpenView University Association, OVUA'97*, Madrid, Spain, April 2–4 1997.

[BLMM 98] Rosy Barruffi, E. Lamma, Michela Milano, and P. Mello. Planning with incomplete and dynamic knowledge via Interactive Constraint Satisfaction. In *Workshop on Integrated Planning, Scheduling and Execution in Dynamic and Uncertain Environments*, Carnegie Mellon University, Pittsburgh, PE, USA, June 7–10 1998.

[FeKu 92] David Ferraiolo and Richard Kuhn. Role-Based Access Control. In *Proceedings of the 15^{th} National Computer Security Conference*, volume 2, pages 554–563, 1992.

[FrKo 98] Svend Frølund and Jari Koistinen. QML: A Language for Quality of Service Specifiation. Technical Report HPL-98-10, Hewlett-Packard Laboratories, Palo Alto, 1998.

[GHW 99] Boris Gruschke, Stephen Heilbronner, and Norbert Wienold. Managing Groups in Dynamic Networks. 1999. subimitted to IFIP/IEEE Integrated Management (IM'99).

[Goh 97] Cheh Goh. A Generic Approach to Policy Description in System Management. In Aruna Seneviratne, Vijay Varadarajan, and Pradeep Ray, editors, *Proceedings of the Eighth IFIP/IEEE International Workshop on Distributed Systems, Operations & Management (DSOM)*, pages 1–12, Sydney, Australia, October 21–23 1997.

[Goh 98] Cheh Goh. Policy Management Requirements. In *Proceedings of the 5th Workshop of the OpenView University Assosiation: OVUA'98*, Rennes, France, April 1998.

[GuBa 97] Barbara Guttman and Robert Bagwill. *Internet Security Policy: A Technical Guide.* National Institute of Standards and Technologie, U.S. Department of Commerce, July 1997. NIST Special Publication 800-XX. Draft. Available from `http://csrc.nist.gov/isptg/`.

[Heil 98] Stephen Heilbronner. Policies in the Management of Nomadic Computing Systems. In *Proceedings of the 5th Workshop of the OpenView University Assosiation: OVUA'98*, Rennes, France, April 1998.

[HoRe 91] P. Holbrook and J. Reynolds. Site Security Handbook. RFC 1244, Network Working Group, July 1991.

[Hosm 91] Hilary H. Hosmer. Metapolicies I. In *ACM SIGSAC Data Management Workshop, San Antonio, TX*, December 1991.

[Hosm 92] Hilary H. Hosmer. Metapolicies II. In *Proceeding of the 15th National Computer Security Conference, Baltimore, MD*, pages 369–378. United States Government Printing Office: 1992-625-512:60546, October 1992.

[KKK 96] Thomas Koch, Christoph Krell, and Bernd Krämer. Policy Definition Language for Automated Management of Distributed Systems. In *The Second International Workshop on Systems Management*, Toronto, Canada, June 19–21, 1996. IEEE Computer Society.

[Koch 97] Thomas Koch. *Automated Management of Distributed Systems.* Berichte aus der Informatik. Shaker Verlag, 1997. Doctoral thesis, FernUniversität Hagen, 1996.

[Krel 95] Christoph Krell. Policy-basiertes Management von verteilten Systemen. Diploma thesis, FernUniversität Hagen, 1995.

[Kühn 95a] Winfried E. Kühnhauser. On Paradigms for Security Policies in Multipolicy Environments. In *Prceedings of the 11th International Information Security Conference (IFIP/SEC'95)*, Cape Town, South Africa, 1995. Chapman & Hall.

[Kühn 95b] Winfried E. Kühnhauser. A Paradigm for User-Defined Security Policies. In *Proceedings of the 14th IEEE Symposium on Reliable Distributed Systems*, Bad Neuenahr, Germany, September 1995. IEEE Computer Society Press.

[KüvKO 95] Winfried E. Kühnhauser and Michael von Kopp Ostrowski. A Framework to Support Multiple Security Policies. In *Proceedings of the 7th Annual Canadian Computer Security Symposium*, Canadian System Security Centre, Ottawa, Canada, May 1995. Communications Security Establishment.

[LeMa 97] Ulf Leonhardt and Jeff Magee. Security Considerations for a Distributed Location Service. HP Open View university association workshop, Department of Computing, Imperial College, London, March 21, 1997.

[LMSY 95] Emil C. Lupu, Damian A. Marriott, Morris S. Sloman, and Nicholas Yialelis. A Policy Based Role Framework for Access Control. In *First ACM/NIST Role Based Access Control Workshop*, Gaithersburg, USA, December 1995.

[LSY 97] Emil C. Lupu, Morris Sloman, and Nicholas Yialelis. Policy Based Roles for Distributed Systems Security. In *Proceedings of the 4th Workshop of the OpenView University Association, OVUA'97*, Madrid, Spain, April 2–4 1997.

[LuSl 97a] Emil C. Lupu and Morris S. Sloman. Conflict Analysis for Management Policies. In Aurel A. Lazar, Roberto Saracco, and Rolf Stadler, editors, *Proceedings of the Fifth IFIP/IEEE International Symposium on Integrated Network Management (IM'97)*, San-Diego, Carlifornia, U.S.A., May 12–16 1997. Chapman & Hall.

[LuSl 97b] Emil C. Lupu and Morris S. Sloman. A Policy Based Role Object Model. In *Proceedings of the First International Enterprise Distributed Object Computering Workshop (EDOC'97)*, Queensland, Australia, October 24–26 1997. IEEE.

[LuSl 97c] Emil C. Lupu and Morris S. Sloman. Reconciling Role Based Management and Role Based Access Control. In *Proceedings of the Second ACM Workshop on Role-Based Access Control*, Fairfax, Virginia, USA, November 6–7, 1997.

[LuSl 97d] Emil C. Lupu and Morris S. Sloman. Towards a Role Based Framework for Distributed Systems Management. *Journal of Network and Systems Management*, volume 5(1), 1997. Plenum Press.

[Marr 97] Damian A. Marriott. *Policy Service for Distributed Systems*. PhD thesis, Imperial College of Science, Technology & Medicine, London, June 1997.

[MaSl 96] Damian A. Marriott and Morris S. Sloman. Implementation of a Management Agent for Interpreting Obligation Policies. In *Proceedings of the Seventh IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM)*, Scuola Superiore G. Reiss Romoli (SSGRR), L'Aquila, Italy, October 28–30 1996.

[MoSl 88] Jonathan D. Moffett and Morris S. Sloman. The Source of Authority for Commercial Acces Control. *IEEE Computer*, volume 21(2):59–69, February 1988.

[MoSl 91] Jonathan D. Moffett and Morris S. Sloman. The Representation of Polices as System Objects. In *Proceedings of the Conference on Organizational Computer Systems (COCS´91)*, pages 171–184, Atlanta, Georgia, November 5–8 1991.

[MoSl 93a] Jonathan D. Moffett and Morris S. Sloman. Policy Conflict Analysis in Distributed System Management. *Journal of Organizational Computing*, April 1993.

[MoSl 93b] Jonathan D. Moffett and Morris S. Sloman. Policy Hierarchies for Distributed Systems Management. *IEEE JSAC Special Issue on Network Management*, 11(9), December 1993.

[MoSl 93c] Jonathan D. Moffett and Morris S. Sloman. User and Mechanism Views of Distributed Systems Management. *Distributed Systems Engineering Journal*, 1(1):37–47, August 1993.

[MSY 95] Damian A. Marriott, Morris S. Sloman, and Nicholas Yialelis. Management Policy Service for Distributed Systems. research report DoC 95/10, Department of Computing, Imperial College, London, United Kingdom, October 1995.

[NIST] National Institute of Standards and Technology. *An Introdution to Computer Security: The NIST Handbook.* U.S. Department of Commerce. Special Publication 800-12.

[PCF 91] R. Pethia, S. Crocker, and B. Fraser. Guidelines for the Secure Operation of the Internet. RFC 1281, Network Working Group, November 1991.

[Radi 98] Igor Radisic. Konzeption eines Policy-basierten Konfigurationsmanagements für nomadische Systeme in Intranet. Diploma thesis, Ludwig-Maximilian-Universität München, August 1998.

[RBP+ 91] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen. *Object-Oriented Modelling and Design.* Prentice Hall, 1991.

[SCFY 96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-Based Access Control Models. *IEEE Computer*, volume 29(2):38–47, February 1996.

[Schi 98] Christian Schiller. Eignung von JMAPI für das integrierte Management. Dipolma thesis, Technische Universität München, August 1998.

[SlLu 98] Morris Sloman and Emil Lupu. Policy Work at Imperial College. Department of Computing, Imperial College London, 1998.

[Slom 94a] Morris Sloman, editor. *Network and Distributed Systems Management.* Addison-Wesly Publishing Company, 1994.

[Slom 94b] Morris Sloman. Policy Driven Management for Distributed Systems. *Journal of Network and Systems Management*, 2(4), 1994. Plenum Press.

[Stan 95] British Standard. BS 7799: Information security management. Part 1. Code of practice for information security management systems, 1995.

[Stan 98] British Standard. BS 7799: Information security management. Part 2. Specification for information security management systems, 1998.

[Vis 97a] Visigenic. *VisiBroker for Java: Programmer's Guide*, September 1997. Version 3.0, Part number: 211-357-00000-1.

[Vis 97b] Visigenic. *VisiBroker: Naming and Event Services Programmer's Guide*, September 1997. Version 3.0, Part number: 211-315-00000-1.

[Wies 95] René Wies. *Policies in Integrated Network and Systems Management: Methodologies for the Definition, Transformation, and Application of Management Policies.* Doctoral thesis, Ludwig-Maximilian-Universität München, 1995.

[WMS 97] René Wies, Maria-Athina Mountzia, and Philip Steenekamp. A Practical Approach Towards a Distributed and Flexible Realization of Policies Using Intelligent Agents. In Aruna Seneviratne, Vijay Varadarajan, and Pradeep Ray, editors, *Proceedings of the Eighth IFIP/IEEE International Workshop on Distributed Systems, Operations & Management*, pages 292–308, Fairfax, Virginia, USA, October 21–23 1997.

[Wood 97] Charles Cresson Wood. *Information Security Policies Made Easy.* Baseline Software, Inc, Post Office Box 1219, Sausalito, Carlifornia 94966-1219 USA, 1997. Version 6.