İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

POLICY ENFORCEMENT

M.Sc Thesis by
Delal NASIR

504001570

Department: Computer Engineering

Programme: Computer Engineering

Supervisor : Prof. Dr. Bülent ÖRENCİK

MAY 2003

ACKNOWLEDMENTS

I would like to express my deep appreciation to Professor Bülent ÖRENCİK for his guidance and tolerance while I was studying on this thesis.

I am grateful to Taner Dursun for his all helps, technical guidance and for the time he spent to answer my questions patiently.

A person owes most of her/his knowledge to the business environment, and I feel very lucky to have such colleague, teaching me so much and making the office a great place. I thank all and especially to our Project Manager Oktay Adalier and my friends Müge Çevik, Cenk Özden, Erkan Aydın, Burçin Bozkurt, Zafer Bayraktar and Şerif Bahtiyar.

Of course, I would like to thank Halil Özçiçek for his existence making the world a better place to live for me.

And finally I would like to express all my gratitude to my family, who gave me all the strength and the courage throughout my life.

May 2004 **Delal NASIR**

TABLE OF CONTENTS

ABBREVIATIONS **VII**

LIST OF TABLES **IX**

LIST OF FIGURES **X**

ÖZET **XI**

SUMMARY **XIII**

1 INTRODUCTION **1**

1.1 Thesis Contributions 3
1.2 Thesis organization 4

2 POLICY BASED MANAGEMENT **5**

2.1 Policy Based Management System Components 6
2.1.1 Policy 7
2.1.2 Policy Management Tool 10
2.1.3 Policy Repository 11
2.1.4 Policy Decision Point 11
2.1.5 Policy Enforcement Point 12
2.1.6 Communication Protocols 13
2.1.7 Repository and Policy Information Models 14
2.1.8 PBM Execution Models 19
2.1.9 PBM Target Domains 20
2.2 A Historical Perspective 22
2.3 Implemented Products 24
2.3.1 Academic Products 24
2.3.2 Commercial PBM Products 27

3 POLICE **31**

3.1 POLICE Policy Definition Language 31
3.1.1 Language Syntax 33
3.1.2 Policy Instantiation 35
3.1.3 Policy Types 36
3.1.4 Constraint Expressions 38
3.1.5 Event Definitions 39
3.1.6 Constant Definitions 41
3.2 POLICE Policy Based Management System Architecture 41
3.2.1 Architecture Overview 41
3.2.2 Architectural Analysis of Framework 44
3.2.3 Policy Distribution 56
3.3 Conflict Handling 56
3.3.1 Conflict Classes 57
3.3.2 Police Conflict Handling Method 58

4 ENFORCEMENT MECHANISM OF DIFFERENT SYSTEMS 59

4.1 Active PBM 59
4.1.1 Active PBM Architecture 60
4.1.2 Policy Enforcement Point architecture 63
4.2 PONDER 64
4.2.1 PONDER System Architecture 64
4.2.2 PONDER Enforcement Architecture 66
4.3 Script MIB 69
4.3.1 Script MIB System Architecture 69
4.3.2 Enforcement (Policy Engine) 72
4.4 HP OpenView PolicyXpert 75
4.4.1 PolicyXpert System Architecture 76
4.4.2 Policy Enforcement Points 77
4.5 IPHIGHWAY 78
4.5.1 OPS System Architecture 79

5 POLICE POLICY ENFORCEMENT POINT 81

5.1 General Architecture of PPEP 81
5.2 PPEP Manager 82
5.3 Policy Controller 83
5.3.1 Policy Controller Components 83
5.3.2 Operation of Policy Controller 85
5.4 Event Service 88
5.4.1 Operation of Event Service 88
5.4.2 Condition Table 90
5.4.3 Policy Gauge 91
5.4.4 Event Service Manager 95
5.4.5 Inter-Node Event Collector 96
5.5 Resource Controller 99
5.5.1 Controller Table 100
5.5.2 Operation of Resource Controller 100
5.6 Policy Deployment Service Agent 101
5.7 Node Management Interface 102
5.7.1 Operation of NMI 104
5.8 DataBase 105
5.9 Messaging Service 105
5.10 Domain Server Registration 105
5.11 Comparison with Other Management Systems' PEPs 106

6 IMPLEMENTATION OF PPEP 108

6.1 JMS 108
6.2 DataBase 109
6.2.1 Table Structures 109
6.3 Simulation of Managed Device and NMI 110
6.4 Evaluation of PPEP 111

7 CONCLUSION 112

REFERENCES 114

APPENDIX A.1 COMPARISON TABLE 119

AUTOBIOGRAPHY **121**

ABBREVIATIONS

AC :Access Controller
AEO :Authorization Enforcement Object
APO :Authorization Policy Object
CIM :Common Information Model
CIMOM :Common Information Model Object Manager
CLI :Command Level Interface
COPS :Common Open Policy Service
DARPA :Defense Advanced Research Projects Agency
DB :Data Base
DEN :Directory Enabled Networks
DMTF :Distributed Management Task Force
EC :Event Consumer
ES :Event Service
FTP :File Transfer Protocol
GPI :General Policy Interface
GUI :Graphical User Interface
HTTP :Hypertext Transfer Protocol
IETF :Internet Engineering Task Force
INEC :Inter Node Event Collector
IPX :Internet Packet Exchange Protocol
ISP :Internet Service Provider
JMS :Java Messaging Service
JVM :Java Virtual Machine
LAN :Local Area Network
LDAP :Lightweight Directory Access Protocol
MIB :Management Information Base
MOF :Managed Object Format
MPLS :Multi Protocol Label Switching
NMI :Node Management Interface
OEO :Obligation Enforcement Object
OPO :Obligation Policy Object
OPS :Open Policy System
ORB :Object Request Broker
PBM :Policy Based Management
PC :Policy Controller
PCIM :Policy Core Information Model
PDP :Policy Decision Point
PDS :Policy Deployment Service
PEP :Policy Enforcement Point
PG :Policy Gauge
PIL :Police Interface Language
PMA :Policy Management Agent
PMC :Policy Management Console

QoS	:Quality of Service
QPIM	:Policy QoS Information Model
REO	:Refrain Enforcement Object
RPO	:Refrain Policy Object
RSVP	:Resource ReSerVation Protocol
SLAs	:Service Level Agreements
SNIA	:Storage Networking Industry Association
SNMP	:Simple Network Management Protocol
TCP	:Transmission Control Protocol
UDP	:User Datagram Protocol
URL	:Uniform Resource Locator
VLAN	:Virtual Local area Network
VPN	:Virtual Private Network
WAN	:Wide Area Network
WBEM	:Web Based Enterprise Management

LIST OF TABLES

Page Number
Table 3.1 Police policy syntax 34
Table 3.2 The syntax for policy templates and instantiations 35
Table 3.3 PDL Events and usage in Police 40
Table 5.1 Policy DB 84
Table 5.2 Condition Table 91
Table 5.3 Controller Table 100
Table 6.1 Policy Controller Policy Table 110
Table 6.2 Condition Table 110

LIST OF FIGURES

Page Number
Figure 2.1 General Architecture for a PBM System 6
Figure 2.2 Management architecture 26
Figure 3.1 Architecture of the Police framework 43
Figure 3.2 Sample statements written in PIL 45
Figure 3.3 Policy Classes hierarchy generated by compiler 46
Figure 3.4 Policy compiler structure 47
Figure 3.5 Example obligation policy 48
Figure 3.6 Generated Java code snapshot 49
Figure 3.7 Predefined classes 50
Figure 3.8 Steps to create a compiler using SableCC 51
Figure 3.9 Sample domain structure 53
Figure 3.10 Model class for a serial port and a relationship 54
Figure 3.11 A graphical representation of sample management schema 55
Figure 4.1 Architecture of Inter-domain A-PBM 61
Figure 4.2 PEP environment architecture 64
Figure 4.3 PONDER Management system architecture 65
Figure 4.4 Overview of policy deployment model 66
Figure 4.5 Policy object states 67
Figure 4.6 Overview of a Policy Management Agent 68
Figure 4.7 Architecture of PBM based on the JASMIN Java runtime engine. 71
Figure 4.8 Architecture of the Script MIB based configuration management. 72
Figure 4.9 Different levels of PDP distribution. 72
Figure 4.10 Two different approaches to PBM with the Script MIB. 73
Figure 4.11 The standard Script MIB runtime engine 74
Figure 4.12 Implementation of the policy runtime engine. 75
Figure 4.13 PolicyXpert architecture. 77
Figure 4.14 Open Policy System (OPS) 79
Figure 5.1 POLICE Management System Policy Enforcement Architecture 82
Figure 5.2 Enforcement of an obligation policy 86
Figure 5.3 Enforcement of an authorization policy 87
Figure 5.4 Block Diagram of ES in Police PEP 90
Figure 5.5 A sample Policy Gauge output 93
Figure 5.6 Event Consumers schema for constraint: 95
Figure 5.7 Request Message 96
Figure 5.8 Reply Message 96
Figure 5.9 Inter Node Event Collector messaging via Inter Node Event Bus 98
Figure 5.10 Block diagram of Resource Controller 100
Figure 5.11 Flow diagram of PDS Agent on PPEP 102
Figure 5.12 CIM Instrumentation of a Managed Node. 103

ÖZET

Bir IP ağı, üzerinde pek çok uygulama çalışan ve çok sayıda kullanıcısı olan çok sayıda yönlendirici ve sunumcudan oluşur. Ağ teknolojileri gelişip daha karmaşık hale geldikçe, servis sağlayıcılar müşteri isteklerini karşılamak ve performans ve maliyet açısından optimize etmek için heterojen ve dağıtık mimariler kurmaktadır.

Bu tür sistemlerin yönetimleri çok iyi ağ yöneticileri için bile son derece zor bir hale gelmiştir. Bugün ağ yöneticileri günlük hayattaki ihtiyaçları makine diline çevirmekle yükümlüdür. Bu cihazların her birini ayrı ayrı yönetmektense, hepsinin anlayacağı ortak bir dil oluşturmak çok daha mantıklı bir çözümdür. Bunu sağlayan sistemlere "Strateji Tabanlı Ağ Yönetim Sistemi (PBM)" denmektedir.

PBM sistemlerinin sunduğu hizmetler, ağ yöneticisi için bir yönetim konsolu, kuralları saklamak için bir veri deposu, kuralları depolanabilir bir hale getiren bir karar merkezi ve yönetilecek düğümler üzerinde bir strateji uygulama noktasıdır.

Bu tez yönetim sistemlerinin uygulama noktaları üzerine yoğunlaşmıştır. Tezin amacı kimi strateji uygulama sistemlerinin uygulama noktası tasarımlarını incelemek ve Police yönetim sistemi ile birlikte çalışacak bir uygulama noktası tasarımı yapmak ve kodlamaktır.

Police kendine ait nesne tabanlı bir dili ve strateji gönderme ve uygulama metodu olan dağıtık bir yönetim sistemidir ve Taner Dursun ve Prof. Dr. Bülent Örencik tarafından İstanbul Teknik Üniversitesi'nde geliştirilmiştir.

POLICE dili güvenlik ve yönetim stratejilerini destekler ve diğer sistemlerden farklı olarak stratejilerin uygulanma metodu her tür strateji için aynıdır. Temel olarak "izin" ve "emir" olmak üzere iki çeşit kural vardır. Literatürde bu stratejiler pozitif ve negatif olmak üzere ikiye ayrılmaktadır ancak POLICE'de basitliği sağlamak üzere bu ayrımdan kaçınılmıştır.

İzin stratejileri sistemdeki bir nesnenin diğer bir nesne üzerinde belirli bir işlemi yapıp, yapamayacağını beliler. Emir stratejileri ise bir nesnenin yapmak zorunda olduğu işlemleri belirtir.

Police bileşenleri şunlardır:

- Strateji sunumcusu: Strateji yönetiminden sorumludur. Strateji nesnelerini saklar ve uygulama noktalarına gönderir.
- Yönetim arayüzü: Ağ yöneticisinin strateji kurallarını belirlemesini ve uç noktaları gözlemlemesini sağlar. Yöneticinin stratejileri doğru yazmasına yardımcı olur.
- Uygulama noktaları: Strateji Kontrol Birimi ile sonlanır ve stratejilerin yönetilen düğümlere gönderilmesini ve orada uygulanmasını sağlar.
- Sistem bilgi modeli: Sistemin bütün bilgilerini depolar.

Police uygulama noktası bileşenleri:

- Strateji Kontrol Birimi: Strateji kurallarını merkezden alır, veri tabanında saklar ve Olay Servisi ile iletişimde bulunarak yönetilen düğümlere gönderir.
- Mesaj Servisi İstemcisi: Sistem içindeki uygulamalar doğrudan haberleşmezler, bir sunumcu üzerinden haberleşirler Police'de Java Mesajlaşma Servisi (JMS) Sunumcusu ve İstemcileri kullanılmaktadır.
- Strateji Gönderme Servis Temsilcisi: Strateji gönderme merkezinden gelen kuralları ilk alan birimdir, bu kuralları daha sonra Strateji Kontrol Birimine gönderir.
- Olay Servisi: Yönetilen düğümlerdeki ve gerekli durumlarda uzak düğümlerdeki olayları toplar ve Strateji Kontrol Birimini bu konuda bilgilendirir.
- Yönetilen Düğüm Arayüzü: Yönetilen birim ile Strateji Uygulama Noktası arasındaki arabirimdir.
- Kaynak Kontrol Birimi: İzin stratejileri için Strateji Kontrol Birimleri ile yönetilen düğümler arasında bir arayüzdür.

SUMMARY

A typical IP network consists of a large number of routers and servers running a variety of applications that are used by a large number of users. As the network technologies become more and more complex the network providers have to merge a set of heterogeneous networking technologies, distributed applications and systems in order to fulfill customer needs as well as optimizing the physical network in terms of performances and costs.

Consequently, the management of such information technology resources becomes increasingly complex to managers even with a very strong skill. Network managers today must translate business objectives into specific operational behaviors within the network infrastructure to ensure that network resources are allocated in accordance with business needs. Configuring a high number of routers, bridges, or servers by generic rules instead of individual configuration appears to be less complex, less error-prone and more flexible. Policy Based Management (PBM) Systems gets the high level rules compiles and send the to the network nodes and enforces these rules at the node.

Policy Based Management Systems are being developed to serve a Policy Management Tool (a Policy Console), a Policy Server (Policy Decision Point - PDP), a Policy Repository and a Policy Enforcement Point (PEP).

This thesis aims to define Policy Enforcement of Policy Based Networks, presents the architecture and implementation of Police Policy Enforcement Point "Police PEP" and gives an analysis of some other PBM systems' PEPs.

Police is a Policy Based Network Management Framework composed of a policy specification language and its deployment and management models for policy distribution in distributed systems. It is developed by Taner Dursun and Prof. Dr. Bulent Orencik at Istanbul Technical University.

Police has an object-oriented language for specifying both security and management policies for distributed systems. Unlike many other policy specification notations, uniform enforcement for all kind of policies makes the Police framework simpler. It uses two basic policy types: authorizations and obligations. In Literature, these two classes can be further divided into positive and negative policies. However, negative policies are eliminated to handle policy conflicts easily. Although negative policies complicate enforcement process, there are reasons to support them.

Authorization (cando) Policies are designed to give permissions to actors to run some actions on other system objects. Obligation (mustdo) Policies define what actions an actor must do if the conditions specified in the policy are met or certain events occur.

POLICE components are:

- Policy Server acts as the interface to policy management, which stores compiled policy classes, creates and distributes new policies and otherwise supports policy management coordination. Policy Management Service

creates deployment objects. Policy Deployment Service (PDS) coordinates the distribution of the deployment objects to the PEPs.

- The Management Console (MC) provides a graphical interface for specifying, reviewing and modifying policies. Being integrated with Domain Service and Police Compiler, the MC helps the administrator to define valid policies conforming to both object model of the whole system and the language syntax.
- Enforcement agents, termed Policy Controllers (PCs) are responsible for enforcement of both authorization and obligation type policies.
- Whole information related to the managed system, stored in Domain Service, is called as System's Information Model (SIM).

Police Enforcement components are:

- Policy Controller: Policy Controller (PC) is the unit which is responsible for the receiving of the policies from the Policy Deployment Service, storing them in the DataBase and communication with the Event Service (ES) for Policy Gauge (PG).
- Messaging Service Client: Rather than communicating directly with each other, the components in an application based around a message service send messages to a message server. The message server, in turn, delivers the messages to the specified recipients. JMS is used for messaging.
- Policy Deployment Service Agent: Policy Deployment Service Agent receives the policy objects sent by Policy Deployment Service via messaging service, and sends them to Policy Controller.
- Event Service: The Event Service collects and composes events from the managed nodes and from Event Services of other PEPs, and forwards them to the registered PCs which are executing policies.
- Node Management Interface: Node Management Interface (NMI), which is a mediation layer, provides a standard interface between the PEP and the embedded hardware and software.
- Resource Controller: Resource Controller is the unit which acts as an interface between the PCs and the managed objects for the authorization policies.
- DataBase used to store the policies.

1 INTRODUCTION

As the IP networks consist of a large number network devices and servers running a variety of applications and users and the network technologies become more and more complex, the network administration is becoming more complex too. Providers have to merge a set of heterogeneous networking technologies, distributed applications and systems in order to fulfill customer needs as well as optimizing the physical network in terms of performances and costs.

Translating business objectives into specific operational behaviors within the network infrastructure is getting harder for the managers even with a very strong skill. Configuring a high number of routers, bridges, or servers by generic rules instead of individual configuration appears to be less complex, less error-prone and more flexible. Policy Based Management (PBM) Systems gets the high level rules compiles and send the to the network nodes and enforces these rules at the node.

A typical PBM system includes a Policy Management Tool (Policy Console), a Policy Repository, a Policy Decision Point (Policy Server) and a Policy Enforcement Point (PEP).

The policy management tool mainly provides a console to interact with the human administrator and allows writing high-level policies; compiles the entered policies, checks the syntax and stores them in a repository.

The policy repository is a data store where the policies are stored by the policy management tool and retrieved by policy decision points or policy enforcement points. The repository provides the central location that drives the operations of the entire network.

Policy Decision Point (also known as Policy Server) retrieves the policies from policy repository and transforms them in format that the policy enforcement points can understand. Then PDP sends the lower level policies to the related PEPs.

The Policy Enforcement Point represents the component that always runs on the policy aware node. It is the point at which policy decisions are actually enforced.

Policy enforcement point is the PBM system component which is responsible for the receiving of policies from PDP and enforcing these policies to the target managed device that the PEP supports.

One PBM system is Police that is composed of a policy specification language and its deployment and management models for policy distribution in distributed systems. It is developed by Taner Dursun and Prof. Dr. Bulent Orencik at Istanbul Technical University.

Police has an object-oriented language for specifying both security and management policies for distributed systems. Unlike many other policy specification notations, uniform enforcement for all kind of policies makes the Police framework simpler. It uses two basic policy types: authorizations and obligations. In Literature, these two classes can be further divided into positive and negative policies. However, negative policies are eliminated to handle policy conflicts easily. Although negative policies complicate enforcement process, there are reasons to support them.

Authorization (cando) Policies are designed to give permissions to actors to run some actions on other system objects. Obligation (mustdo) Policies define what actions an actor must do if the conditions specified in the policy are met or certain events occur.

POLICE components are:

- Policy Server acts as the interface to policy management, which stores compiled policy classes, creates and distributes new policies and otherwise supports policy management coordination. Policy Management Service creates deployment objects. Policy Deployment Service (PDS) coordinates the distribution of the deployment objects to the PEPs.
- The Management Console (MC) provides a graphical interface for specifying, reviewing and modifying policies. Being integrated with Domain Service and Police Compiler, the MC helps the administrator to define valid policies conforming to both object model of the whole system and the language syntax.
- Enforcement agents, termed Policy Controllers (PCs) are responsible for enforcement of both authorization and obligation type policies.

Whole information related to the managed system, stored in Domain Service, is called as System's Information Model (SIM).

This thesis is interested in policy enforcement of management systems; it analyzes different enforcement mechanisms and presents the Policy Enforcement Point architecture and implementation of Police Policy Based Network Management Framework.

1.1 Thesis Contributions

In this thesis, policy enforcement of management systems is studied.

Some well known PBM systems, Ponder, Script MIB, Active PBM, HP OpenView and IP HighWay OPS are researched and enforcement mechanisms of these systems are analyzed.

A Policy Enforcement Point is designed and implemented to be interoperable with the Police PBM system. This policy enforcement point is named PPEP (Police Policy Enforcement Point).

PPEP is compared to the other systems' enforcement points which are researched. PPEP includes the following components:

- Policy Controller: Policy Controller (PC) is the unit which is responsible for the receiving of the policies from the Policy Deployment Service, storing them in the DataBase and communication with the Event Service (ES) for Policy Gauge (PG).
- Messaging Service Client: Rather than communicating directly with each other, the components in an application based around a message service send messages to a message server. The message server, in turn, delivers the messages to the specified recipients. JMS is used for messaging.
- Policy Deployment Service Agent: Policy Deployment Service Agent receives the policy objects sent by Policy Deployment Service via messaging service, and sends them to Policy Controller.
- Event Service: The Event Service collects and composes events from the managed nodes and from Event Services of other PEPs, and forwards them to the registered PCs which are executing policies.

- Node Management Interface: Node Management Interface (NMI), which is a mediation layer, provides a standard interface between the PEP and the embedded hardware and software.
- Resource Controller: Resource Controller is the unit which acts as an interface between the PCs and the managed objects for the authorization policies.

Node management interface of PPEP is not completely designed, and it is only simulated in the implementation. The detailed architecture of node management interface is being developed in Sakarya University by Hasan Er.

1.2 Thesis organization

Chapter 2 describes general architecture of Policy Based Management and main components of a PBM system as well as the aims of PBM and different approaches to some concepts.

Chapter 3 aims to describe "Police" which is a PBM Framework designed by Taner Dursun as Ph. D. Thesis. Police Policy Definition Language and architectural analysis of Police is given in this chapter.

In Chapter 4 Enforcement Mechanisms of different management systems are studied. These are PONDER, Script MIB (Management Information Base), Active PBM, HP OpenView PolicyXpert and IPHighway Open Policy System (OPS).

In Chapter 5, design of Police PEP, the Policy Enforcement Point of Police, is given in detail and a comparison of Police PEP with the PEPs of the systems described in Chapter 4, is included.

Chapter 6 gives the details and results of implementation.

2 POLICY BASED MANAGEMENT

A typical IP network consists of a large number of routers and servers running a variety of applications that are used by a large number of users. Some of these applications (such as the applications implementing the routing protocols) are needed for proper operation of the IP networks, and the other applications, such as Web servers or mail servers, are deployed for a specific purpose. Although individual configurations of the IP network vary from organization to organization, it is safe to assume that the different applications and users in the network interact with each other in a complex and subtle manner. The applications compete with each other to access network resources, such as the bandwidth on different links, the buffer space in routers, encryption facilities for secure communication, processing cycles at servers, etc.

As the network technologies become more and more complex the network providers have to merge a set of heterogeneous networking technologies, distributed applications and systems in order to fulfill customer needs as well as optimizing the physical network in terms of performances and costs.

Consequently, the management of such information technology resources becomes increasingly complex to managers even with a very strong skill. Network managers today must translate business objectives into specific operational behaviors within the network infrastructure to ensure that network resources are allocated in accordance with business needs. It is not practical to control the network infrastructure manually, so configuring a high number of routers, bridges, or servers by generic rules instead of individual configuration appears to be less complex, less error-prone and more flexible. The rising technology which gets the policies, creates these generic rules, stores and enforces them is called “Policy Based Management”.

Policy based management is an administrative approach that is used to simplify the management of a given strategy by establishing policies to deal with situations that are likely to occur. Policies are operating rules that can be referred to as a means of

maintaining order, security, consistency, or other ways of successfully furthering a goal or mission [1].

In the computing world, policy based management is used as an administrative tool throughout an enterprise or a network, or on workstations that have multiple users or network devices like routers, firewalls, Virtual Private Networks (VPNs) etc. Policy based management includes policy based network management, the use of policies to control access to and priorities for the use of resources. Policy based management may be used in systems management, or the creation and operation of an efficient computing environment.

2.1 Policy Based Management System Components

A typical PBM system includes a Policy Management Tool (Policy Console), a Policy Repository, a Policy Server (Policy Decision Point) and a Policy Enforcement Point (PEP), as shown in **Figure 2.1**. The Policy Server talks to the PEP through a protocol such as Common Open Policy Service (COPS), Simple Network Management Protocol (SNMP) etc. The data processed by a PBM is the policy. Policies are business objects translated into sets of rules that are represented as data structures [2].

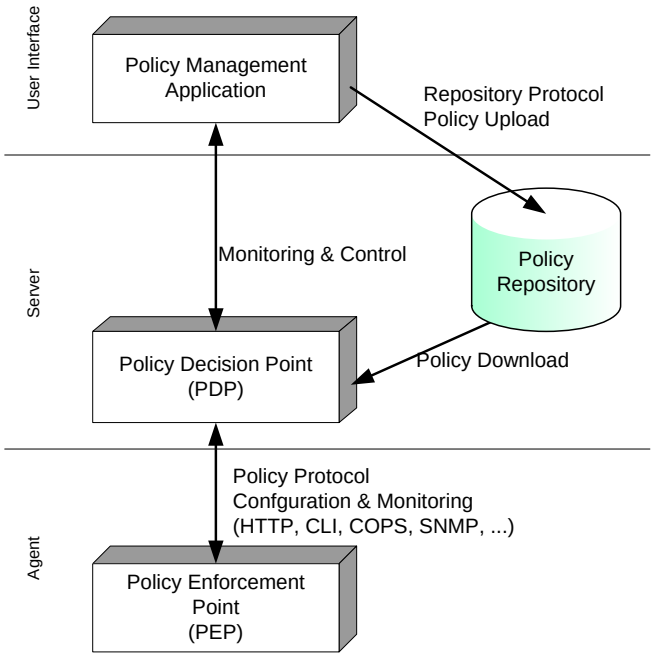

Figure 2.1 General Architecture for a PBM System

2.1.1 Policy

There are a lot of definitions for policies from academic and commercial communities. According to IETF RFC3198 "Policy" can be defined from two perspectives:

- A definite goal, course or method of action to guide and determine present and future decisions. "Policies" are implemented or executed within a particular context (such as policies defined within a business unit) [3].
- Policies as a set of rules to administer, manage, and control access to network resources [4].

Storage Networking Industry Assoc's (SNIA) defines policy as:

- "The measurable, enforceable and realizable specification of method, action and/or desired state that meets service requirements in a storage-based information infrastructure."
- A means of extending the functionality of management systems *dynamically*
- Guides the behavior of a network or distributed system through high-level declarative directives
- dynamically introduced, checked for consistency, refined and evaluated
- resulting typically in a series of low-level actions

In order for policies to be active in the network, an administrator needs to define them, and the devices need to enforce them. For a smooth operation of the network, the policies must satisfy several requirements [5]:

- Precision. A policy must be precise and specified to a level of detail that can be understood by a network element. There should be no ambiguity regarding a policy's interpretation. From the point of view of a network element, the policy must be specified with all the requisite details that might be needed for the enforcement of a policy.
- Consistency. The set of policies that is given to a network element or to all the network elements in the network must be consistent. For example, if two

firewalls are encrypting secure traffic among themselves, they must be configured to use a set of algorithms and keys that both can use to interoperate.

- Compatibility. The set of policies that is given to a network element must be compatible with the capabilities that the network element can support. For example, many routers in the network look at the network layer header in the packet, so the policy must be specified to such a network element in terms that include a network layer header field.
- Mutual consistency. If multiple policies are specified for a network element, they must be consistent with each other. The network element should be able to apply these policies in an unambiguous manner. The set of policies should not specify a conflicting action be taken. At the very least, if conflicting policies are found, the network element should be able to determine unambiguously the policy that should be selected among all the possible choices.
- Ease of specification. In order for a network administrator to specify policies, a policy must be simple and easy to specify. A network operator must be able to specify the policies with the minimum effort required to specify a consistent and precise set.
- Intuitiveness. In order for the network operator to specify policies, policy definition must be done in terms that are familiar to the network operator. These terms might require that policies be defined in terms of the business processes or concepts that are commonly used within an organization. A network operator might find it much easier to specify a policy in terms of an application, such as payroll processing, rather than in terms of the machine addresses and header fields that are used by the payroll application.

It is clear that all these requirements cannot be optimally satisfied together simultaneously. For a specification that is intuitive, the policy must be specified in terms of familiar business concepts. However, the compatibility requirement states that just the opposite to be done –that is, the policy must be specified in terms that can be handled by the routers and servers in the network. Similarly, ease of use requires that much of the detail of the operation of the network elements be hidden

from the network operator. This is in direct contrast to the preciseness requirement [5].

The reason for this apparent conflict is that there are at least two different levels of policies:

- One that a human operator would like to enter and specify
- One that the devices would actually be capable of enforcing

Each of these levels imposes a different set of requirements on the policy definition and specification. The human operator uses a higher-level policy to express his/her objectives, which are then translated into a lower level of policies that are interpreted by each of the devices.

Low level policies are specified to match a device's capabilities and include device specific commands. High level policies are more likely to human language sentences and depend on the end goal of the network operator. High level policies may be specified in a format that is intuitive to human operator A human operator must be able to specify the policies without significant effort.

2.1.1.1 Rule

A policy is defined as a combination of one or more sets of rules and rules are the components of policies that define specific criteria, or conditions, that when evaluated to TRUE, trigger certain actions.

Each rule binds an action to sets of conditions that describe who (users), what (systems, applications), and in what circumstances the actions may be triggered. PBM rules can be broadly grouped into three categories: actions, conditions and roles. The details of each part are explained below.

- Actions**:** Actions specify the behavior that managed devices must follow. Actions are typically associated with the handling of requests for services or capabilities provided by one or more hardware or software components within a network systems environment.
- Conditions**:** Conditions allow manager to specify under what circumstances specific behavior is to be in effect. Conditions can be given as a single event, a logical combination of events or a case waiting for a certain value of a certain module in the system.

- Role: At the point of PBM, role is an administratively specified characteristic of a managed element (for example, an interface). [3]

2.1.2 Policy Management Tool

The policy management tool mainly provides a console to interact with the human administrator and allows writing high-level policies; compiles the entered policies, checks the syntax and stores them in a repository. It needs to perform the following functions:

- Present a user interface to the human administrator in order to enter the policies in a more user friendly level than the level that the managed devices can understand.
- Validate the syntactic and semantic correctness of user input and any relationships that must exist between the different parameters.
- Detect any conflicts between the high level policies.
- Validate that the policies can be satisfied by given the network's resources and constraints. For example, a policy that asks that a traffic flow be encrypted using DES algorithm cannot be satisfied if the firewalls in the network do not support that algorithm.
- Validate that the policies specified cover all the important scenarios that might be required for a specific application. This especially important for the policies related to security.
- Determine which policies are applicable to which set of policy targets.
- Determine which low-level policies can be used to support the high-level policies specified by the user and transform the high level policies in a format that they can be stored in policy repository.
- Store the policies in the policy repository ant detect when the stored policies are modified. In the case of a modification, the management tool may choose to notify the relevant policy consumers of a change in the policies using a notification mechanism.

2.1.3 Policy Repository

The policy repository is a data store where the policies are stored by the policy management tool and retrieved by policy decision points or policy enforcement points. The repository provides the central location that drives the operations of the entire network.

An example of a policy repository is an LDAP directory that contains both the high-level and low-level policies required for network operations. Instead of a directory, a database server or a Web server can be used. In a Web server, different files represent the policy information stored for the various devices and for each low-level device specific policy.

Regardless of the type of repository selected, the policy stored in the repository needs to conform to specific conventions. These conventions are specified as the information model for policy specification.

2.1.4 Policy Decision Point

Policy Decision Point (also known as the Policy Server) retrieves the policies from policy repository and transforms them in format that the policy enforcement points can understand. Then PDP sends the lower level policies to the related PEPs.

The PDP is responsible for the following functions:

- Locating the set of rules that is applicable to any PEP it is managing, and retrieving the rules from the repository.
- Transforming the set of rules it retrieves from the policy repository to a format and syntax that is understood by the PEP function.
- Decide the PEP related to the policy that is to be sent to the managed device.
- Checking the current state of the network and validating that the conditions required for the application of any policy are satisfied.
- Keeping track of the changes in the policies that might take place by monitoring the repository or by listening to any notification from the management tool.
- Detect any policy conflicts.

The entity implementing the policy would not be able to determine which action to perform. The implementers of policy systems must provide conflict detection and avoidance or resolution mechanisms to prevent this situation. "Policy conflict" is contrasted with policy error. Policy errors occur when attempts to enforce policy actions fail, whether due to temporary state or permanent mismatch between the policy actions and the device enforcement capabilities.

2.1.5 Policy Enforcement Point

The PEP represents the component that always runs on the policy aware node. It is the point at which policy decisions are actually enforced [6]. Policy enforcement point is the PBM system component which is responsible for the receiving of policies from PDP and enforcing these policies to the target managed device that the PEP supports.

The basic interaction between the components begins with the PEP. The PEP will receive a notification or a message that requires a policy decision. Given such an event, the PEP then formulates a request for a policy decision and sends it to the PDP. The request for policy control from a PEP to the PDP may contain one or more policy elements (encapsulated into one or more policy objects) in addition to the admission control information (such as a flowspec or amount of bandwidth requested) in the original message or event that triggered the policy decision request. The PDP returns the policy decision and the PEP then enforces the policy decision by appropriately accepting or denying the request. The PDP may also return additional information to the PEP which includes one or more policy elements. This information need not be associated with an admission control decision. Rather, it can be used to formulate an error message or outgoing/forwarded message [6]. This kind of execution is based on outsourcing model.

Another approach is provisioning model. In provisioning model PEP do not send a policy request to PDP, PDP itself makes a policy decision and sends it directly to PEP needing no requests from PEP. The PEP also sends information to PDP (or management server according to the application) about the status of policy deployments and the configuration states of the devices supported by that PEP. The PEP also sends acknowledgments to PDP for any policy operation that is directed by PDP. This is practical for identifying errors that occur while trying to install the

policy, or for detecting failures that could have an effect on a previously installed policy decision.

2.1.6 Communication Protocols

In a PBM system communication is a subject between Policy Management Application, Policy Decision Point and Policy Enforcement Point. The protocols such as COPS, SNMP, Command Level Interface (CLI) [5], Telnet [7], XML [8], CORBA [9] and LDAP [10] are used for these communications. Some protocols are described below.

SNMP is a framework (including a protocol) for managing systems in a network environment. It can be used for policy-based configuration and control using a specific MIB Module designed to execute policies on managed elements via scripts. The elements (instances) in a network device are evaluated using a policy filter, to determine where policy will be applied [3].

COPS is a simple query and response Transmission Control Protocol (TCP) based protocol that can be used to exchange policy information between a Policy Decision Point (PDP) and its clients (PEPs). The COPS protocol is used to provide for both the outsourcing and the provisioning of policy. Another usage is for the provisioning of policy [3].

COPS has been developed by the IETF Resource Allocation Protocol (RAP) WG as a policy protocol for use in PBM systems. COPS represents a revolutionary approach to the proactive management of network devices. It was developed as a reaction to traditional network management protocols, such as SNMP, which were found to be incapable of efficiently supporting Policy Based Networks. COPS is becoming the protocol of choice over SNMP, which offers reasonable device-monitoring capabilities. COPS may be used for the configuration of several different types of network services such as Diff-Serv Routing, Multi Protocol Label Switching (MPLS), Security, VPN, and VoIP.

The IETF concentrates on the protocols to be used between different components with most of the efforts focusing on LDAP for storing policies, and the common open policy service (COPS) protocol for communication between a PDP and a PEP.

2.1.7 Repository and Policy Information Models

There are integration problems among current management systems, especially in data specification. Common-problems encountered in data specification can reduce the reuse of management data. These problems are:

- The same data can be named differently,
- The same data may be realized using different data types or could be organized differently in each repository.

If the data are named and/or organized differently in each management system, or stored in different data types, sophisticated mediation software will need to be used to enable exchange of the data between different management systems.

Policy repository is a specific data store that holds policy rules, their conditions and actions, and related policy data such as logical container for reusable policy elements. A database or directory would be an example of such a store. Directories and LDAP are the industry choice for interoperable standard policy storage.

An information model is "an abstraction and representation of the entities in a managed environment -their properties, operation, and relationships." This is independent of any specific repository, application, protocol, or platform. Information models have been designed for use with policy protocols in PBM to exchange policy information among the PBM components.

The Storage Networking Industry Association (SNIA) has announced the Common Information Model (CIM) Object Manager, designed to help create interoperable and well-managed storage systems and services. It provides an open source implementation of the Distributed Management Task Force's (DMTF) Web-Based Enterprise Management (WBEM) standard.

Vendors can use the CIM Object Manager to define and manage standard objects, such as computer systems, storage media, and network ports, and their interrelationships, in an enterprise. The CIM Object Manager operates as a service layer, interfacing management agents and data providers to client applications. A multi-organizational effort involving the DMTF (Distributed Management Task Force), the IETF, and others, has been defining both an information model and the LDAP [10] schema including CIM, Directory Enabled Networks (DEN) and Policy

Core Information Model (PCIM), to represent standard policy information. Information model is an abstraction and representation of the entities in a managed environment, their properties and operations, and the way that they relate to each other. It is independent of any specific repository, application, protocol, or platform.

2.1.7.1 Lightweight Directory Access Protocol

Lightweight Directory Access Protocol (LDAP) is a standard protocol and API used to access (read, write and query) Directory Services. The use of a directory as the central repository for a PBM system has several issues (performance, replication, and notifications) [10]. LDAP is the most common way vendors access stored user and resource data, and store policy information in these proprietary directories, though its application differs by vendor. LDAP directory can be used as a distribution and/or interoperability mechanism as well as a common directory format for sharing policies between multiple PBM systems.

The PBM architecture assumes that multiple policy systems may need to interoperate within a single domain, and share the same policy information. A central repository can be used to store, distribute, and coordinate policy information among such systems.

2.1.7.2 Common Information Model

CIM [11,12] is an object-oriented information model which is developed by the DMTF for describing management data. An information model requires a set of legal statement types or syntax to capture the representation, and a collection of actual expressions necessary to manage common aspects of the computer systems. The CIM captures notions that are applicable to all areas of management, independent of what technology is implemented. The CIM schema maintained by the DMTF includes models for Systems, Applications, Local Area Networks (LAN) and Devices. The CIM schema enables applications from different developers on different platforms to describe management data in a standard format so that it can be shared among a variety of management applications. It consists of a specification detailing the abstract modeling constructs and principles of the Information Model, and a textual language definition to represent the Model. CIM's schemas are defined as a set of files, written in the language of the Specification. Sets of classes and associations represent CIM's Core and Common Models, defining an information

model for the "enterprise" - addressing general concepts, and systems, devices, users, software distribution, the physical environment, networks and policy. The xmlCIM Encoding Specification defines XML elements, written in Document Type Definition (DTD), which can be used to represent CIM classes and instances. The CIM Operations over Hypertext Transfer Protocol (HTTP) specification defines a mapping of CIM operations onto HTTP that allows implementations of CIM to interoperate in an open, standardized manner.

2.1.7.3 Policy Core Information Model

PCIM [11,12] is an information model describing the basic concepts of policy groups, rules, conditions, actions, repositories and their relationships was proposed by IETF. This model is described as a "core" model since it cannot be applied without domain-specific extensions (for example, extensions for Quality of Service (QoS) or IPsec). PCIM is "core" with respect to the area of policy.

COPS protocol requires a new data model called the Policy Information Base (PIB). The PIB is an information model proposed for use with the COPS protocol, to describe policies and the format of policy information exchanged between the PEP and PDP.

2.1.7.4 Directory Enabled Networks

The policies in the directory are typically at a level of abstraction that is distinctively business centric. In addition, this information is typically static relative to the policy rules in the PEPs, which need to be installed and changed in order to implement the business policies

DEN [13] is an information model and schema produced by a consortium of vendors (such as Cisco, Microsoft) in order to promote the concept of application-aware networking through directory-based data integration. DEN allows defining a general directory schema that may include every kind of data on the network (such as users, devices, applications). DMTF develops the specifications of CIM related to the DEN. The work of the DEN group became the input for the IETF Policy WG and for a series of enhancements to the DMTF Common Information Model (CIM).

The original goal of DEN was to define a set of network services that met the needs of policy based systems. DEN provides a set of abstractions that enable the services that applications need to be matched to the services provided by network. The

current network management approaches are not enabling to use business rules to develop management systems. On the other hand, the policy-based systems have not been fully realized yet. DEN is proposed to solve these two problems by translating business rules into device configuration.

2.1.7.5 WEB Based Enterprise Management

WBEM [14] is a set of management and Internet standard technologies that provides a unified management of enterprise computing environments across multiple vendor environments. WBEM simplifies system management, providing better access to both software and hardware data that is readable by WBEM compliant applications. WBEM has been designed to be compatible with all the major existing management protocols. The DMTF has developed a core set of standards that make up WBEM, which includes a data model, the Common Information Model (CIM) standard; an encoding specification, xmlCIM Encoding Specification; and a transport mechanism, CIM Operations over HTTP.

WBEM implementations use a client – server model. The client applications generate requests against a Common Information Model Object Manager (CIMOM) using the WBEM client application programming interfaces (API). The CIMOM may interpret these requests using information in its internal store or support SNMP, CMIP and proprietary protocols through the use of Providers. These are Java translator classes responsible for acting as an interface between the CIMOM and the real managed object. The provider uses HTTP to interact with the CIMOM above and whatever protocol is appropriate to interact with the managed object below. Consequently, the provider classes are responsible for detecting and forwarding events to the management applications. WBEM Services are explained below:

- CIM Object Manager (CIMOM)**:** listens for WBEM client requests for CIM operations possible on CIM objects (ex. create/modify/delete class/instance, etc.). Requests on CIM class definitions are handled directly by the CIMOM. Requests on instances of a CIM class are dispatched to the Provider responsible for extracting the dynamic data for that CIM Class.
- CIM Repository: is used by the CIMOM to store CIM class definition and dynamic data.

- Managed Object Format (MOF) Compiler: converts CIM class definitions in a text MOF file into CIM Repository formatted data.
- MOF-to-JavaBeans Generator: generates JavaBeans based on the CIM class definitions in a text MOF file.
- CIM Workshop: graphical application allows to browse the CIM schema and to perform operations on CIM Classes and Instances.

A CIM object is a computer representation, or model, of a managed resource, such as a printer, disk drive, or CPU. CIM objects are imported into a CIMOM.

The CIMOM provides a central repository for CIM objects on a WBEM-enabled system where WBEM clients in a network can go gather information about managed resources within the system. The CIMOM transfers information among WBEM clients, the CIMOM-Repository, and managed resources. When a WBEM client application requests information about a managed resource, the CIMOM contacts either the appropriate provider responsible for the CIM object that represents that managed resource or the CIMOM-Repository. When a WBEM client application requests data from a managed resource that is not available for the CIM Object Manager Repository, the CIM Object Manager forwards the request to the provider for that managed resource. CIMOM will use schema information to determine which providers to contact.

Providers are special classes that communicate with managed objects to retrieve and then forward data to the CIMOM for integration and interpretation. Providers do the followings:

- Register with the CIMOM - Providers register themselves with CIMOM by specifying a Provider Qualifier. Classes and properties marked by the provider qualifier will be an indication to the CIMOM that the associated information is dynamic and must be obtained from the providers rather than the repository.
- Provide data to management applications – If the requested data is not available in the CIMOM-Repository, the CIMOM forwards the request to the provider responsible for that managed resources. On request generated by CIMOM, the provider accesses the data from the managed resource and passes the data back to the CIMOM. If the data received from a managed

resource is in a native format (such as C code), the provider maps the data to Java CIM classes prior to passing it to the CIMOM. Providers serve dynamic information to the CIMOM.

- Control management resources - When a management application sends data to the CIMOM to control a managed resource, the CIMOM passes the data to the appropriate provider. If the managed resource requires data in a native format, the provider maps the CIM classes to the resource's native format prior to passing it along

The requests originated by management applications may be for static information such as schema definitions or static instances that is stored in the CIMOM Repository. For the static data, the CIMOM should route the request to the repository.

2.1.8 PBM Execution Models

The PBM architectures can be classified as Outsourcing and Provisioning Policy Models according to the operation flows between entities. Another classification can be made according to the place of the enforcement logic and the classes are Active Policy and Active Agent Models. These classes can also be used in an integrated manner.

In the case of an outsourcing policy model, the PDP receives policy requests from a network device (PEP), and determines whether or not to grant these requests. A PEP issues a query to delegate a decision for a specific policy event to another component, external to it. For example, a PEP requires a fast policy decision by sending a query to the PDP, asking for a policy decision to determines whether or not to grant permission to applications. Here, the PDP either accepts or rejects the request of applications based on the business-level policy found in the repository. This model is sometimes referred to as "Pull" mode, or "reactive" mode, since the PEP pulls policy decisions from the PDP, while the PDP responds according to the PEP events.

In the policy-provisioning model where network elements are pre-configured based on policy, prior to processing events, the PDP predicts future configuration needs, and proactively provisions resources accordingly or some new policies entered at the management console might be distributed in real-time. In other words, rather than

responding to PEP events, the PDP prepares and "pushes" configuration information to the PEPs.

"Outsourced Policy" is contrasted with "Provisioned Policy", but they are not mutually exclusive and operational systems may combine the two. Both models employ policy servers as the PDP to control the network devices that enforce the policy (i.e. PEPs).

In Active Policy Model [15,16], the policy rules are written in a protocol independent, platform independent language, the policy rule can execute in a PEP as an active policy, when the policy is deployed. The management functionality is delegated to autonomous agents such as active network capsules [17] and Mobile Agents [18] operating on behalf of the PDP and PEP. Active PBM uses this approach.

Active Manager (Passive Policy) concept uses policies as passive objects; these policies do not instigate management operations. Managers which are generally stationary software agents are the active objects which are responsible for interpreting a policy and performing the activities specified. Most of the work adopt this approach such as Ponder [19].

Script MIB PBM system presents both Active Policy and Active Management concepts.

2.1.9 PBM Target Domains

Generally, PBM can be used almost for any domain, including Security and QoS management domains. In addition, policy has also been proposed for Routing management. While policy can describe constraints on all these service domains, the operational constraints on these domains differ and these differences can influence the tradeoffs made in implementing a policy-based management system.

The security domain (filtering and cryptography) is the least forgiving of error. If there is a hiccup in the enforcement of a security policy, the connectivity may be permanently loosen or the trust may be loosen by allowing intruders access. The policy applications in these domains generally include Access Control, Role Based Access Control applications.

Routing policy has the biggest scaling problem. Huge numbers of routers must be coordinated to consistently enforce the policy goals. For this reason, routing policy tends to be more dynamic and tolerant of changes in the routes. Routing domain policy applications are usually concerned with the routers, routing protocols and policy servers controlling them.

QoS policies provide abstractions to simplify the management of QoS mechanisms. They contain rules that govern how resources can be used, or how applications and users should be treated. Policies form a bridge between Service Level Agreements (SLAs) and the network elements that deliver the desired performance and services. SLAs between customers and their service providers define what QoS should be and how much it will cost the customers to get it.

QoS policy enforcement falls somewhere, between the security domain and the routing domain. An edge-oriented QoS administrative model will be on the same scale as firewall enforcement. A more detailed administrative model will impact more enforcing devices and have more of a scaling concern.

The rise of PBM has resulted from the growing need for automated QoS management. Organizations are managing their existing bandwidth using a QoS-focused PBM product. The fit between QoS and PBM is natural. QoS introduces unfairness, some applications get preferential treatment at the expense of others. Policy rules are an excellent vehicle to express and control this desired unfairness.

Recent work in the IETF has led to the development of standards for a QoS-enabled Internet. Through the efforts of the Integrated Services working group, data flows can describe their desired or required service to the network. Resource ReSerVation Protocol (RSVP) carries the service requests into the network, where the acceptance of these QoS requests results in better network service to some flows, possibly at the expense of service to traditional best-effort flows.

Integrated Services (IntServ) is a standard framework designed to address the need for better than best effort service in an IP network in which voice, video, and data have been integrated. IntServ provides the ability for applications to choose among multiple, controlled levels of delivery service for their data packets. It requires that network elements support the establishment of these service levels, and that applications have a mechanism to signal their requirements (e.g. RSVP).

Differentiated Services (Diff-Serv) is a framework for providing differentiated classes of service for Internet traffic. A bit-pattern (known as a code-point) marks an IP packet to receive a particular treatment, or per-hop behavior, at each network node. The IP header field, called the DS-field in IPv4, defines the layout of the ToS (Type of Service) octet; in IPv6 it is the Traffic Class octet. Differentiated Services requires policy to define the correspondence between code points in the packet's DS-field and individual per-hop behaviors. Routers do not need to maintain state for each flow. Instead, they provide the differential service based solely on the edge packet marking.

Resource ReSerVation setup Protocol (RSVP) is a QoS signaling protocol developed by the IETF RSVP Working Group to establish a reserved "pipe" of Integrated Services (Int-Serv) reservations end-to-end. RSVP is a standard, signaling protocol for applications to request and reserve bandwidth across a routed network. The IETF has begun work on using RSVP to combine the benefits of IntServ and DiffServ.

Multi-Protocol Label Switching (MPLS) [20] is new IP-forwarding paradigm using labels attached to packets, enabling classification, traffic engineering (explicit routes), and increased scalability/performance. Like DiffServ, MPLS is growing in stature within the IETF and the industry.

Role-based Access Control Management Framework for management of distributed systems [21] is another target domain. In this framework, organizational structure is often specified in terms of organizational positions such as regional, site or departmental network manager, service administrator, service operator. Specifying organizational policies for people in terms of role-positions rather than persons, permits the assignment of a new person to the position without re-specifying the policies referring to the duties and authorizations of that position. The tasks and responsibilities corresponding to the position are grouped into a role associated with the position which is essentially a static concept in the organization. A role is thus the position, the set of authorization policies defining the rights for that position and the set of obligation policies defining the duties of that position.

2.2 A Historical Perspective

Policy work in the industry has been occurring in various consortia and standards bodies through the years. The first use of the term “policy” in the context of IP

networks was for specifying routing policies in the internet. The internet is composed of many subnetworks, each independently administrated by various Internet Service Providers (ISPs) and interconnected via various exchange points. A scheme for describing what types of packets each ISP would accept forwarding was needed at the exchange points. This led to the specification of routing policies and a language for specifying routing policies.

The next step in the area of policies was done in the context of resource reservation policies. This work began within the IETF in 1996 and led to the development of a protocol called "Common Open Policy Service (COPS)". Soon, the developer realized that this could be used for more functions than the original objective. Since late 1998, there has been a lot of activity to insulate COPS from its close ties to resource reservations, and to use it as a general purpose policy protocol.

The biggest boost to policy activity was obtained through the "Directory Enabled Networks (DEN)" Initiative in 1998, led by Microsoft and Cisco corporations. With the dominant position of Microsoft in the workstation operating system market and that of Cisco in the internet router space, this initiative aroused a lot of interest within the industry. The goal of DEN is to build a networking infrastructure that is easier to operate and manage than the traditional ones. The ease of management was obtained by driving the networks from a central repository originally based on a directory.

The basic approach of DEN was to store different types of information related to different network devices and applications in a directory and to use that to manage the network. The major thrust of DEN is to define a standard format (or a schema) by which the properties relating to network devices are defined. In other words, the information stored in the directories had to correspond to a specific model that DEN specified [22].

In order to get a more open standard, the companies involved in DEN decided to migrate the work to the DMTF by the end of 1998. DMTF is an industry consortium formed in 1992 by several computer companies with the goal of developing a common method for managing personal computers. DMTF defines a common information mode that originally intended to be used to describe the characteristics of a computer. It was extended to apply to networks of computers and other devices.

These specifications form the Common Information Model (CIM) for systems and network management.

At the same time that the DMTF activity started, an activity within the IETF was started with the goal of defining a framework that could be used to specify policies across various disciplines, and to show that it could be used in the context of QoS disciplines [5].

2.3 Implemented Products

2.3.1 Academic Products

Ponder framework [23] is one of the academic tools consisting of a policy specification language, an architecture for deploying policies based on the language and a set of tools for specifying and managing policies. In conjunction with the language, the toolkit permits integrated administration of resources, people and policy information with automated policy deployment. The toolkit comprises an Integrated Development Environment (IDE) with a policy compiler, as well as tools for managing policies and roles at runtime. It includes a domain browser provides a common user interface to select policies and enforcement components from the directory, as well as with the policy compiler to interactively instantiate policies. The implementation includes a policy administration toolkit that supports the specification and management of policies. Policies and roles are created using a *policy editor*, compiled and stored in the domain service

Developed in the snmpconf working group of the IETF, [24] includes a tool based on Script MIB and combines the IETF specifications for policy based configuration management and infrastructure for distributed management by delegation. The IETF Script MIB allows sending management scripts containing either policy objects (IETF format) or policy codes (java classes) to the distributed Script MIB agents acting as PDP, which includes a runtime engine designed to evaluate policies. The policy runtime engine contains implementations of all application specific libraries such as PCIM policy classes that are to be evaluated, and it has access to the network elements to be configured by these policies. This tool provides capabilities to transfer management scripts to distributed managers, initiate, monitor, and terminate the execution of management scripts. The evaluation of rules is always initiated by Events. In the future, they plan to replace their language with a policy specification

language such as Ponder. The authors have implemented a limited conditions evaluation mechanism supporting only time-related triggering services. Also, they haven't taken into consideration of conflict detection and resolution yet.

[25] describes a management architecture based on the IETF framework for the policies specified for QoS. In this system, the similar modules as our Police model such as Event Service as Event Manager and PolicyGauge as Sensor are employed.

They use Ponder language as high-level policy language with refinement into expert system rules. Sensors are classes which are used to collect, maintain, and process a wide variety of attribute information within the instrumented processes. Each sensor is associated with an attribute (**Figure 2.2**). The sensor's methods (probes*)* are used to initialize sensors with threshold and target values and collect values of attributes. Probes are embedded into process code to facilitate interactions with sensors. When an applications starts up, it instantiates its "coordinator". The coordinator then registers with the "Name Server". The Name Server receives and maintains registration data collected from other components and application processes in a repository. The Name Server coordinates the interaction between the QoS management components and the application's coordinator.

After the application has registered with the Name Server, it requests its QoS requirements from the Policy Manager. The Policy Manager is the only type of PDP to receive events directly. Corresponding event identifiers are sent to PDP and trigger the execution of rules. The PDP then registers its interest in the event identified by the event identifier with the Event Manager. The PEP has two components. The first component, implemented in Java, interacts with PDPs. The second component has specific "resource managers" such as the CPU manager and the Memory Manager. These resource managers actually manipulate the resources and are specific for applications. There is a resource manager for each resource. The policies were put into XML and references to the XML file were placed in the LDAP directory. Ponder policies are expressed with XML language.

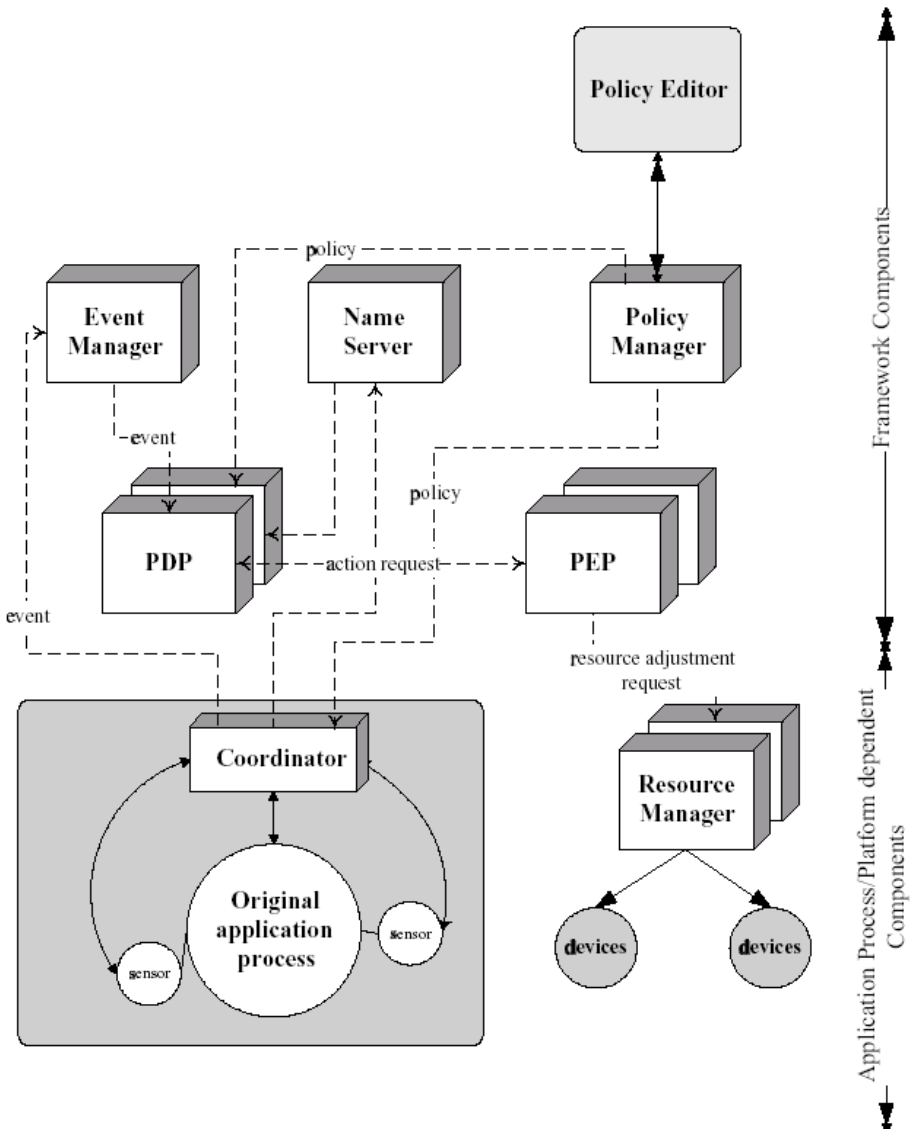

Figure 2.2 Management architecture

[26] describes an architecture and algorithms for on-line discovery of quantitative models without prior knowledge of the managed elements. It is interesting because of its usage of CIM Object Manager (CIMOM) [27] concept and on-line discovery of the managed node's element schema specified with CIM that describes the capabilities of the resources of nodes(e.g., a database has tables and tablespaces), especially associated metrics (e.g., rows read, sort times) and configuration parameters. Each managed node has a CIM agent which is responsible for maintaining current values of the resources in the managed node as well as responding to requests to query the element schema of the managed node. The CIM agent actually is a CIM Object Manager based on the Web based Enterprise Management (WBEM) [27] framework. Responsible for handling incoming requests

and dispatching them to providers, this agent is an SNIA (Storage Networking Industry Association) CIM Object Manager [28]. There is a manager that communicates with the agents of managed nodes to collect data from managed nodes and passes to its clients, management applications. The data requested from Agent are retrieved by the CIMOM, either from its instance repository or directly from the CIM providers.

Verma [29] describes a QoS tool used to specify Service Level Agreements (SLAs) and to manipulate SLA related information in a tabular format. The tool transforms high-level policy information into device configurations, and stores them in an LDAP directory.

2.3.2 Commercial PBM Products

Most of the tools come from industry and is based on the IETF policy framework. The majority of the commercial tools are specific to QoS management, but many also include access control management. Beyond a few basic "must-have" components, PBM solutions tend to vary from one vendor to the next. A common feature in commercial tools is a graphical user interface which typically allows the administrator to visually select a network device or other managed element from a hierarchically arranged tree-view of policy targets, and specify the policies in the form of *if* <*condition*> *then* <*action*> rules for the selected targets. The different products allow the specification of varying degrees of conditions in policy rules including a number of time attributes, source or destination IP addresses, IP type service, TCP and User Datagram Protocol (UDP) port numbers, as well as higher level user-defined data, and allow the user to permit or deny traffic based on those conditions.

The different standards protocols are implemented at varying degrees from the different vendors. A lot of the products support COPS as the main communication protocol for policy information between the components of their architecture, while others support HTTP or CLI for the configuration of routers and switches. In addition, not all vendors support LDAP for storing policies although they use directories as a major component of their products both for storing policy rules as well as network and user information, in order to enable scalability and third-party interoperability. Support for security is also available in many of the commercial

products, and includes access control configuration for firewalls and routers, Unix and Windows operating systems, as well as databases or for web-access. Some products such as Tivoli's and Computer Associates are focusing on enterprise-level management of security for e-commerce applications and support role-based management of user access rights.

Vendors are reinventing network management, transforming its role from passive network monitoring to active QoS and network SLA provisioning. Policy management is necessary to administer network QoS, Security and Resources throughout the enterprise. More than 15 vendors are preparing PBM solutions that promise to enhance QoS and enterprise end to end security [30,31].

The major commercial PBM products are Nortel Optivity policy services, Orchestream, HP Openview PolicyXpert, Cisco CiscoAssure policy networking, Cisco Policy Manager, Allot communications NetPolicy, IPHighway open policy system, Lucent technologies RealNet rules, SolSoft Visual Policy Management for Network Security, Novell ZENworks, Computer Associates Infrastructure Management and eTrust solutions and Tivoli Management Framework product suite.

Device configuration can be accomplished only by employing a combination of CLI (command-level interface), SNMP, COPS and LDAP. But, most vendors plan to implement COPS and LDAP in their products, appropriate to the IETF standards.

There are other roles for PBM, too: For example, while most vendors consider policy management a means to manage device configuration, there are others, such as HP, that see PBM as an end-to-end tool for COPS-enabled desktops as well as network hardware. By pushing policy to the desktop, the network administrator can control applications and set up differentiated services at their source.

In short time, Directory-enabled networks (DEN) will be used to reconcile the user and resource information in directory server with the PBM. Most vendors see a DEN PBM solution in their futures, but how the vendors will use the directory differs greatly. Extreme Networks sees the directory as a way for policy servers to share information, allowing scalability and third-party interoperability. For Allot, using a directory enables its customers to insert new policies without having to develop a special API. Lucent sees the directory as a giant storehouse of information, where each user has a private subtree. For Lucent, the directory is the single sign-on for

complete network resource management, both voice and data services. The vendor plans to accomplish this via a directory-independent LDAP schema. Lucent is the only vendor to implement an LDAP client on their switches. Lucent has plans to integrate a COPS agent and a proxy translator for CLI and SNMP management in a future release of RealNet Rules.

Long term, Cisco plans to build a feedback system so powerful that the network will be able to reprovision itself based on changing network conditions. Future products will use directories as an interface to outside-world data. Allot's NetPolicy was the only solution that featured an active feedback mechanism, which lets the policy administrator view, in real time, the effects of deploying a particular policy on the network. Because all network traffic passes through the Allot hardware, administrators can track flows and the policy's effects on those flows. It allows QoS policies.

HP's OpenView PolicyXpert told in 4.4 [32] is based on COPS and RSVP protocol, which was poorly supported by the rest of the vendors. The policy conditions used in PolicyXpert's can be complex combinations of one or more of the following parameters: time, date, day of week, application type (TCP or UDP port number), source or destination IP addresses or ranges, IP type of service, HTTP Uniform Resource Locator (URL) or Virtual Local area Network (VLAN) ID. In addition, PolicyXpert allows permit or deny traffic based on the RSVP admission control model. Its biggest strengths are its multivendor nature, its RSVP capabilities, which are rivaled only by IPHighway's solution told in 4.5, and its use of COPS to provision all devices on the network. When COPS is not available on a target platform, a proxy agent acts on behalf of the Xpert software to configure the non-COPS-compliant device. PolicyXpert has no filtering or security support, nor is there any LDAP integration. PolicyXpert's is planned to be integrated with the HP OpenView architecture. HP could offer a solution capable of assessing and monitoring service-level guarantees in addition to provisioning policy. PolicyXpert supports only a single policy and deploying it to heterogeneous devices throughout the network, across elements from multiple vendors. Any managed object supporting one or more of the QoS methods can be integrated with PolicyXpert.

The Spectrum Management PBM Suite offers the largest range of conditions of any product, including ethertype (protocol), source or destination physical port, IP ToS

information, IP protocol type, Internet Packet Exchange Protocol (IPX) Class of Service, IPX packet type, IP source and destination addresses or address ranges, IPX network numbers and addresses, IP UDP or TCP port number, and IPX socket number. Spectrum has ground-up directory architecture. The suite implements the latest CIM draft specifications in which to store its policies. LDAP extensions allow the directory to notify the policy manager of changes to the directory.

3 POLICE

POLICE Policy Based Management System is the Ph. D. Thesis of Taner Dursun and is the result of the Taner Dursun's participation in work at TÜBİTAK UEKAE on distributed systems management over the last 5 years. Many of the ideas developed in this thesis are the result of group discussions with Professor Bülent Örencik and our colleagues working in UEKAE.

POLICE develops and presents a complete policy based management framework, which includes a policy specification language and a policy deployment architecture. The several aspects of the framework are already presented in [33-36].

In this section, POLICE PBM System architecture, POLICE policy definition language, system's main components and Conflict Detection mechanism are described.

3.1 POLICE Policy Definition Language

To facilitate policy management activities it is necessary to use a policy language easily understandable by human administrators in conjunction with an integrated toolkit for the deployment, enforcement and coordination of policies within the system. The versatility of policy for use in system management depends a lot on the descriptive power of the policy language. This section describes the policy specification language employed in the Police PBM framework that can be used for specifying general-purpose management policies. Unlike many other policy specification notations, providing uniform enforcement for all kind of policies makes the Police simpler. Another novelty provided by Police is integrating condition and event concepts used separately in Literature. The semantics of the language provide an unambiguous description for the execution of the various elements of the policy specification and can enable further work in policy analysis.

According to Police philosophy, an effective management of a system can be performed by getting activities of the active entities (actor) under control by:

- Arranging access permissions in the system. This can be achieved by defining all permitted actions for each actor (Permission-based regime). An alternative of this approach might be defining both permitted and forbidden actions per actor or per passive objects (targets, which are affected by the activities of the actors.) basis. But this could increase the complexity of the enforcement process that run counter to our goals.
- Arranging the activities of the actors as responses to changing conditions in the system (events, objects properties, etc.). This can be achieved by defining actions must be performed by specific actors in case of condition changes.

Police is declarative, and borrows features from the object-oriented world. The language is flexible, expressive to cover the wide range of requirements implied by the current distributed systems paradigms. In this thesis, however, we adopted some of them but didn't consider several of them as explained in the following list:

- Authorization and obligation policies are involved. Any request made by an actor which is an active object can be defined in terms of an action on an object, so authorization policies define the relationship between actors and actions on target objects.
- Only positive policies are adopted to eliminate some kind of policy conflict as explained in the Section 3.3.2.
- In order to support for monitoring, logging and auditing of events in the managed system, the event constraints are employed not only in obligation policies but also in authorization policies.
- The management actions that need to be performed periodically, or when triggered by events are defined. This is achieved through event constraints in the obligation policies.
- Policies are independent of the implementation of the managed system such as access control mechanisms, operating systems, object-middleware, databases and programming languages.
- Delegation policies specification not adopted to cater for temporary transfer of access rights.

- Constraints are employed in order to restrict the validity of the policies that can be specified in the system under certain conditions based on the state of the system, or the state of the policy targets.
- It is possible to specify policies relating to large systems by employing domain concept to groups of objects.
- Policy reuse and parameterization of policy specifications are adopted.
- The self-management by treating policies as objects in order to enable the specification of policies whose targets are other policies is not yet considered.
- But its object-oriented feature of the language may provide extensibility and scalability automatically.
- Policies are represented flexibly.
- Policies are represented in an unambiguous way.
- Police model support multiple devices and vendors because of the separation between the language and the underlying implementation of the managed system.
- Language is also abstract not to need updates to software on the managed system except the node management interface.
- Language does not support negative policies. However it does not include support to avoid the application specific conflicts. Instead, these types of conflicts are handled by the deployment model employing a distributed mechanism.

3.1.1 Language Syntax

All policies are related to objects with interfaces defined in terms of methods by using an interface definition language so that the policies establish a relationship between the actors that perform the operations and the target objects on which the operations are performed. The term actor refers to users, principals or automated manager components, or any entity initiating operations within the system, which have management responsibility (human or automated manager components). The actors access target objects (resources or service), by invoking the methods are

defined on the interface of the target objects. Managers that are actors for a set of policies may in turn, be target objects for a set of policies.

The notation is essentially aimed at specifying policies which are interpreted by automated agents, but can also be used to specify high-level abstract policies or goals that could be interpreted by humans. The policies are interpreted rather than compiled into the code of agents, so can be changed dynamically. Although Police language does not need syntactic analysis because of the elimination of modality conflicts, it still has a clear syntax that the policies can be analyzed for any purpose. Everything in **bold** is a language keyword in the figures presenting the syntax, including symbols. Choices are enclosed in round brackets ( ) separated by |, optional elements are specified with square brackets [ ] and repetition is specified with braces { }. The complete syntax of the language is written in SableCC [37] source file format (based on BNF). SableCC can be used to specify LALR(1) grammars which are a subset of LR(1) grammars and thus a subset of the context-free grammars that can be specified using BNF.

Table 3.1 Police policy syntax

```
//Authorisation policy
inst cando policyName {
   [actor domainScopeExpression;]
   target domainScopeExpression;
   action action-list;
   when constraintExpression;
}
```

```
//Obligation policy
inst mustdo policyName {
   [actor   domainScopeExpression;]
   target domainScopeExpression;
   action obligationActionList;
   when constraintExpression;
}

//constraint expression
( (always|never)|
  (timeExpression|conditionExpression|eventExpression)* )
```

The syntax of a Police policy is shown in Table 3.1. The actor of a policy, defined in terms of a domain scope expression, specifies the human or automated managers and agents to which the policies apply. The target of a policy, also defined in terms of a domain scope expression, specifies the objects on which actions are to be performed. The actions specify what must be performed for obligations and what is permitted for

authorizations. It consists of method invocations and may list different methods for different object types. Multiple actions in an authorization policy indicate the set of actions or operations which are permitted. Multiple actions in an obligation policy imply that they are performed sequentially after the policy is triggered. The constraint, defined by the when clause, limits the applicability of a policy, e.g. to a particular time period, or making it valid after a particular date (e.g. when time >1/June/1999). In addition, the constraint could be based on attribute values of any managed object in the managed system. Constraints are evaluated automatically and continuously by Event Services as explained in the Section 5.4. Whereas in Ponder, the constraints are evaluated every time an obligation policy is triggered or authorization policy is checked to see whether the policy still applies as attribute values may change. In the managed system, there may be actions executed automatically by hidden actors such as the system's itself (Impersonal Policy). Hence, the actor fields are optional. The action part can be extended to include any kind of statements composed of method calls on local or remote target objects. It specifies what must be performed for obligations and what is permitted for authorizations. Constraints sections can be specified to limit the applicability of both authorization and obligation policies based on time, values of the attributes of the objects, or events.

3.1.2 Policy Instantiation

In addition to direct declaration of policy instances using the syntax shown in **Table 3.2**, the Police language supports reuse of the policy types by instantiating them with different parameters in order to create new policies. Multiple instances can then be created and tailored for the specific environment by passing actual parameters. **Table 3.2** shows the syntax for policy types and direct declaration of policy instances using the keyword "*inst*". Multiple instances can then be created and tailored for the specific environment by passing actual parameters.

Table 3.2 The syntax for policy templates and instantiations

```
//type syntax
type ( cando | mustdo ) policyTypeName ( formalParameters ) {
    [actor  domainScopeExpression;]
    [target domianScopeExpression;]
    action actionList ;
    when constraintExpression ;
}
//instance created
inst (cando|mustdo)policyName=policyTypeName (actualParameters);
```

3.1.3 Policy Types

There are two types of Police policies: authorization policies which specify what activities an actor is permitted to perform on a set of target objects and obligation policies which specify what activities an actor must do to a set of target objects. In the Literature, these two classes can be further divided into positive and negative policies [38,39]. Administrators often express high-level access control in terms of both positive and negative policies; retaining the natural way people express policies is important and provides greater flexibility. Negative authorization policies, for example, can be used to temporarily remove access rights from actors if the need arises. Although there are reasons to support negative policies and many researchers such as [40] have acknowledged their usefulness, they complicate enforcement process by causing several difficulties such as modality policy conflicts. Although this adds the need to analyze policies for conflict detection, this kind of conflict may indicate potentially unforeseen problems with the specification. Instead of negative policies, Police uses constraint keywords like "never" and, "always" to restrict some actions. Actually Police philosophy is based-on "Permission-Based Regime" in which prohibition is the default unless explicitly permitted. The keywords specified above can be used only to narrow scope of policies in terms of actors. Thus, a manager can make an enforcement of a policy differentiate for an individual (actor) than actual actors set.

3.1.3.1 Authorization policies

Authorization (candо) policies are designed to give permissions to actors to perform a set of activities (method calls) on a set of target objects. They are essentially access control policies to protect resources from unauthorized access but are conceptually enforced near the actors instead of target objects to be effected by the operations actor performed. This point differentiates Police from Ponder language. The following example gives a view of the language:

```
type cando Login (actor A, target T, validaTI, int maxLogOff) {
  action T.login();
  when Times.within(validTI) and A.logoffCount <= maxLogOff;
}

inst cando loginPolicy1 = Login(/student/c401,LabServer/server1,
0900-1700,10);
```

```
/* all students from c401 are permitted to login to server named
server1, but only between 09:00 and 17:00 and maksimum 10 times a
day.*/
```

Elements of a policy can be specified in any order. The name of a policy can be specified as a path, thus identifying the directory path into which the policy will be stored.

3.1.3.2 Obligation policies

Obligation (mustdo) Policies define what actions an actor must do if the conditions specified in the policy are met or certain events occur. The obligation policies do not depend on authorization policies. This means that if the conditions for an obligation policy are met, the action will be performed immediately without checking if there is any authorization policy allowing the actor to run this action

Some actions specified in an obligation policy may be related to the real-world entities like human users, another device, etc. We called such objects as Asynchronous Entities (AE) because of indeterministic nature of their behaviors. So, policies about them are actually applied to the software agents representing them in the system. Thus, all obligation policies for a synchronous actor are enforced directly, for an AE indirectly through its representing agent, MOR (Managed Object Representative). MOR's can act as proxies for the user to allow access to resources permitted by the obligation policies or they can act as automated agents, executing obligation policies on the AEs' behalf.

In the following policies, the users must change their passwords when expired and the account failing to login tree times must be disabled.

```
// disable a user account if the user fails 3 times to login
inst mustdo LoginFailure {
   actor s= /Student/Admin;
   target t= /Student/users;
   action t.disable(),s.log(t);
   when 3*loginfail(t);
}

// a password older than one mount is changed
type mustdo PasswordExpiration (actor A){
    target t = A;
    when t.ageOfPwd >= Time.day(30);
    action t.changePwd();
}
```

The actor element is optional as obligation actions may be internal to the system, whereas authorization actions always relate to an actor. The action can be prefixed with the name of the object on which the action is called. If no prefix is specified, the action is assumed to be on the target interface by default. An action within an obligation policy may result in an operation on a remote target object. This could fail due to remote system or network failure so an "exception mechanism" should be provided for obligations to permit the specification of alternative actions to cater for failures which may arise in any distributed system. In Police system there is no such a support yet, whereas Ponder has.

3.1.4 Constraint Expressions

An important element of each policy is the set of conditions under which the policy is valid. This information must be explicit in the specification of the policy. A subset of the Object Constraint Language (OCL) [41] is used to specify constraints in Ponder. OCL is simple to understand and use, and it is declarative – each OCL expression is conceptually atomic and so the state of the objects in the system cannot change during evaluation. The constraints limit the applicability of a policy and are expressed in terms of a predicate, which must evaluate to true for the policy to apply. Policy constraints can be considered as conjunctions of basic constraints, which can be either time or state based. The analysis of a set of policies can then be substantially improved since time-based constraints can be compared for possible overlap and state based constraints can be either simultaneously satisfied or mutually exclusive if they relate to states of the same system component. Limiting the applicability of a policy, constraint expressions can be based on a particular time period, attribute values of objects, or events:

- Actor/target state constraints based on the object state as reflected in terms of attributes at the object interface.
- Action/event parameters constraints based on event parameter values in obligations or action parameter values in authorizations or refrains.
- Time constraints which specify the validity periods for the policy. A time library object provided with the Ponder language is used to specify time constraints.
- Events: simple or complex event expressions.

In contrast to the many other policy languages [42,43], all policies specified in Police can be event-triggered. Events can be real, i.e. internal system events, or pseudo events notified by Event Service components.

In Ponder, however, the events cannot be used in the constraint expressions. Instead, only in obligation policies there is separate element to specify on which event the policy will be triggered. Police provide event constraint for the authorization policies as well by putting event expression into common constraint expression syntax for each type of policies.

3.1.5 Event Definitions

Event-condition-action rules prove to be a very flexible approach to specifying management policy as exemplified by Policy Definition Language (PDL) [44], a language implemented and used in Lucent switching products. In the PDL schema, each set of event instances occurring "simultaneously" in an event stream is called an epoch which is an application specific time window. Primitive events can be composed to form composite events that enable policies to be enforced under any of the following situations. In order to express system events, complicated event expressions are adopted from the PDL, defining several operators to derive composite events from primitive events. Events in Police are used as a Constraint section element to limit the applicability of the policies. Events can be simple, i.e. an internal system event, or a pseudo event notified by changes in the properties of managed objects. Composite events can be specified using event composition operators. Table 3.3 specifies how the event composition operators of the PDL are used in Police. All event operators have equal precedence and evaluation is strictly left to right.

It is possible to define events separately, and reuse them in multiple policies. The definition of an event may be parameterized. Event parameters map onto the event attributes, which define new names within the scope of the policy object where the event is specified. These names can then be referenced within the policy:

```
event timerA = Timer.at("2001:12:15", "22:17:00");
event databaseFailure(db,try)=(connectionLost(db)-> failedSql(try));
```

The first parameter corresponds to the parameter of the connectionLost(db) while the second to the parameter of failedSql(try). The two events that are used in the event expression are assigned to the new event.

Table 3.3 PDL Events and usage in Police

PDL statement	Meaning
e_1| e_2|…| e_n	occurance of one of events **e_i** in an epoch
	Police : **e_1| e_2** (occurs if either **e_1** or **e_2**)
e_1 & e_2 &... & e_n	Simultaneous occurance of **n** events in an epoch
	Police : e_1 && e_2 (occurs if both **e_1** and **e_2** occur)
!e	No occurrence of event in an epoch
	Police : **!{ e_1← e_2 + time-period}** (no occurance of **e_1** during the time-period after the occurance of **e_2**
e_1, e_2	event **e_2** immediately follows an event **e_1**
	Police : **e_1 → e_2** (occurs if **e_1** occurs before **e_2**)
^e	a sequence of zero or more occurrences of an event
	Police : ^e : (at least once occurance of **e**)
group (**e_1 & e_2 &... & e_n**) or **group** (**e_1| e_2|…| e_n**)	a *single event* grouping all instances of the complex event. If there are *n* instances of **e_1** and *m* instances of event **e_2** in an epoch, there is only one occurrence of **group** (**e_1** & **e_2**) instead of *m x n* occurrences
	Police: not adopted
Aggregation functions to evaluate values over multiple epochs.	
Count (e)	Number of occurrences of event **e**. For example, in a sequence of epochs **{ e_1 , e_3 , e_1}, { e_2 , e_3 }, { e_4 }, { e_2, e_1 }, { e_2 }** Count(**e_1,** ^(**e_2 & e_3**)) is equals one and Count(**e_1,** ^(**e_2** | **e_3**)) is 3.
	Police : **n*e** : means the event e occures n times consequently, instead of using "**Count(e)>n**". (Occurs if **e** occurs **n** times).
Other aggregations	**Sum**, **Avg**, **Max**, **Min**, etc.) relating to event parameters to aggregate values over multiple epochs
	Police: not adopted

In addition to ordinary system events, the pseudo-events in the constraint section of a policy are created to observe attributes of the managed system's objects and occur when these attributes reach at specified value ranges. The sophisticated time period conditions can also be implemented as pseudo-events. For example, the statements like "object1.attribute2 > 200", "Date > 10.12.2003" are considered as pseudo-

events. Together with the system events, the pseudo-events compose complex event statements in the constraint section of the policies. To connect the pseudo events with system events, the same operators as specified in Table 3.3 are used.

3.1.6 Constant Definitions

Constants can be defined in Police. A type identifier can be used to indicate the user-defined type of a constant if it is not one of the predefined types (int, real, string, boolean). "User-defined" types are policy types. The "set" type defines a domain scope expression, which can be used to specify actor, target and constraint elements.

```
constant definition =
 int {identifier = expression ;} |
 real {identifier = expression ;} |
 string {identifier = expression ;} |
 boolean {identifier = expression ;} |
 set {identifier = domain_scope_expression ;}

// examples of defining constants.
 int maxLogin = 5;
 string x = managerX.getName();
 string str1 = "this is a string";
 set targetSet1 = /subnetA/routers;
```

Any of the constants described above, including the constraints and events described earlier, can be used to parameterize policy types.

3.2 POLICE Policy Based Management System Architecture

The deployment architecture supports the instantiation, distribution and life-cycle management of policies, as well as their enforcement by automated enforcement components. Police framework presents an implementation of the architecture and an integrated policy-administration toolkit.

3.2.1 Architecture Overview

This section describes general POLICE PBM architecture. Figure 3.1 illustrates the architecture of the management system. It includes three supporting services: a domain service, a policy service and an event service. The policy management service acts as the interface to policy management; it stores compiled policy classes, creates and distributes new policy objects. The "domain service" manages a distributed hierarchy of domain objects and supports the efficient evaluation of actor

and target sets at runtime. Each domain includes managed objects but also references to the PEPs and policy objects that currently apply to that domain.

In concept, the domain service is similar to a directory service such as LDAP, with extensions to allow changes to the membership of a directory to be distributed to interested parties, e.g. via events. The "event service" collects and composes system events as well as those from the managed objects in the system, and forwards them to registered policy controllers to enforce policies. The event service exposes a publish/subscribe interface whereby policy controllers or other event services can subscribe to receive certain types of events.

The user (i.e. policy administrator) interacts with the policy and domain services to design the domain structure, specify new policies, compile and store them in the domain service. Management tools interface with the two services and provide a graphical environment for specifying and managing policies. The distribution of policies is initiated through the policy management service. Policy Objects (POs) are generated and maintained by the policy service to manage policies at runtime. This includes the distribution of the policies to their enforcement components, and the handling of dynamic domain-membership changes to the objects to which the policies apply. The fact that policies explicitly define their actors and targets makes the automated distribution of policies possible. All kinds of policies are distributed to their actors.

The entities that enforce policies are called Policy Controller (PC) and are distributed. PCs implement a policy enforcement interface to support loading, unloading, enabling and disabling of policy objects.

Resource Controller intercepts the authorization requests from actor objects and forwards to the PCs in order perform checks on whether the access is permitted. PCs typically do not have to interface to lower-level mechanisms that really carry out the actions; for example a firewall protecting the services on its network, an operating system protecting its resources, or a database manager protecting its databases. A PC normally orders all the actors at its location and enforces all kinds of policies relating to them.

The actors of a policy can be any objects that initiate invocations. PCs directly enforce all types of policies for an actor, and their architecture can be generic,

although they can be targeted to specific applications e.g. QoS, security or storage management. The enforcement of policies requires a PC to subscribe the event specifications with the event service. The event service notifies the PC of events which trigger its policies, and the PC is then perform the executing the actions defined by those policies if the constraints are evaluated as true.

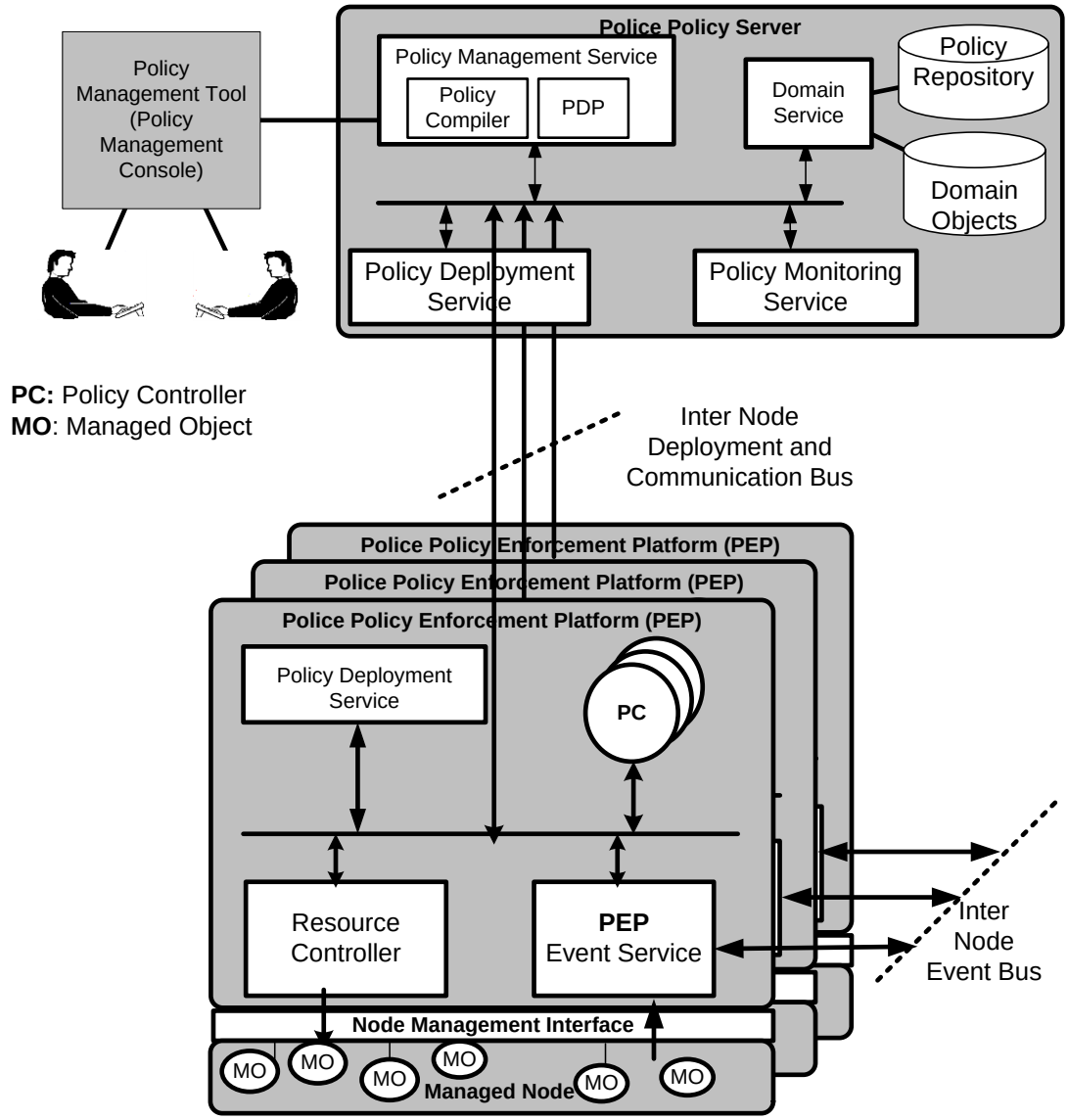

Figure 3.1 Architecture of the Police framework

3.2.2 Architectural Analysis of Framework

3.2.2.1 Policy Server

The Police Policy Server acts as the interface to policy management, which stores compiled policy classes, creates and distributes new policy objects and otherwise supports policy management coordination. Policy Management Service creates a deliverable Deployment Object for each policy to be distributed. Policy Deployment Service coordinates the distribution of the deployment objects to the related PEPs'.

The Policy Management Console (PMC) provides a graphical interface for specifying, reviewing and modifying policies. Being integrated with both the Domain Service and the Police Compiler, the Management Console helps the administrators to define valid policies conforming to both information model of the entire system and the language syntax. Existing policies and policy types can be selected with the aid of the PMC, modified, recompiled and redistributed to enforcement points.

By monitoring errors, events, conflicts, and the decisions made about conflicts at runtime, the administrator also uses the PMC to interact with the enforcement operations performed through the entire system. The design of the PMC also includes an Integrated Development Environment (IDE) tool allowing policy-programming. By using it, the administrators can create and modify policies of the system as if they write a code to be run on a distributed system. As relying on the Police language, the interface language of this IDE will support loop, if-else statements, and many other structures of an ordinary computer programming language. The language is called as Police Interface Language (PIL), since its statements are interpreted into Police policies. In the Figure 3.2, sample PIL statements are shown.

A Command-Line Window is also provided to enter single-line commands for the Policy Management Server. This allows interactive management and monitoring of policy enforcement process. The PIL commands entered through this window will affect the enforcement process immediately. A "live interface" to whole system will be realized for the administrators to manipulate the system.

```
//...
UserObject user1 = new UserObject(tdursun, Taner, DURSUN,...);
DomainService.addElement ("\user\student", user1);
actor actor1 = (Actor) user1;

when (host1.noUserSession()){
  actor1 cando host1.method1();
  actor1 cando host1.method3();
}

for  (Target  t  in [target1, target2, ..., targetn]){
  actor1 cando  t.method2() when ;
}
//...
```

Figure 3.2 Sample statements written in PIL

3.2.2.2 Policy Compiler

Policy Compiler is responsible for generating Java class for each policy defined through the PMC. Then, these Java classes are put in a deployment object and sent to enforcement platforms by the Policy Deployment Service.

The fact that the language is high-level and platform independent, it imposes the design of a runtime representation for policies to enable distribution to the enforcement components, and make the interpretation of policies easy. Policies are enforced by automated management agents, and thus the enforcement code for these policies must be suitable for interpretation or execution by distributed enforcement components. A very attractive solution for the enforcement codes is the use of Java code, which can be distributed across networks and executed on any system with a Java virtual machine.

One alternative is to generate a Java program for each of the policies specified, which can then be loaded into any agent for execution. The Java code provides interaction with the underlying event and domain services in order to enforce the policy. The generated Java program will be used in the same way Java mobile code is used to extend the functionality of the agents in the system in a plug-and-play fashion. Although this approach has some disadvantages such as security risks, it is attractive and the research on this type of implementation is a part of POLICE future work.

The POLICE compiler is implemented in Java and a default Java code generator is included to generate a Java object for each policy in order to satisfy the above requirement. The compiler has been integrated with a policy editor that simplifies policy specification.

A policy is implemented in the runtime system as an object; the policy compiler generates a Java object which maintains a data-structure representation of each policy and stores it in the domain service. A policy object can then be accessed from the domain service and distributed to the Police PEPs (enforcement points responsible for enforcing that policy), and it enables other objects and components in the system to query it to determine the actor, target and other elements of the policy. The representation of policies as runtime objects follows the class hierarchy presented in Figure 3.3. The policy compiler creates an appropriate subclass from the class hierarchy for each of the user-defined policy types.

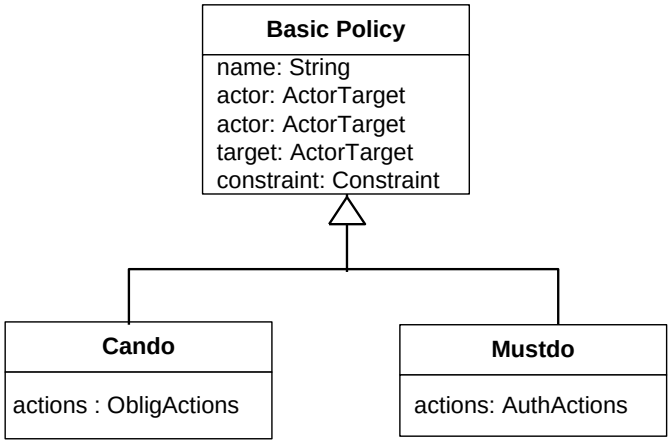

Figure 3.3 Policy Classes hierarchy generated by compiler

Figure 3.4 shows the main modules of the policy compiler framework. The main phases of the compiler generate intermediate code, which is then passed on to the code-generator added to the compiler. The code assembler module is responsible for storing the generated code for a given policy in the domain service under the appropriate domain entry. Its implementation is thus specific to the domain service implementation. The default representation of a policy is the Java code. The components of the compiler implementation are illustrated in Figure 3.4.

The compiler is based on a LALR(1) parser generated with SableCC [37], an object-oriented Java parser by building a strictly-typed “abstract syntax tree” that matches the grammar of the language, automatically generates “tree-walker” classes and

enables the implementation of the actions on the nodes of the tree using inheritance. The SyntaxAnalyser, generated with SableCC using the grammar of the language parses a policy specification into an abstract syntax tree (AST) of Java objects. The AST is then passed on to the semantic analyzers. Semantic analysis is performed by handling the definition of names (scope analysis) and checking that policies contain all the required elements (completeness checks). After the completion of the scope analysis, type analysis is performed, and an intermediate code (IC) is generated. The IC is a tree structure of Java objects corresponding to the policies of the specification. The IC is passed on to the CodeGenerator which is enabled by the user to generate code. The code generator calls the Policy Service to store the generated code in the directory. A Java code generator generates a Java object representation of policies. Policy types are compiled into Java policy classes and stored in the domain hierarchy. Instantiation of a policy type creates and initializes a Java policy object. Predefined classes exist which implement the basic functionality for each of the classes corresponding to policy types. User-defined policy types are generated as Java classes which extend the appropriate basic policy class. For basic policies that are specified directly as instances, an unnamed Java class is generated in the background, and then instantiated.

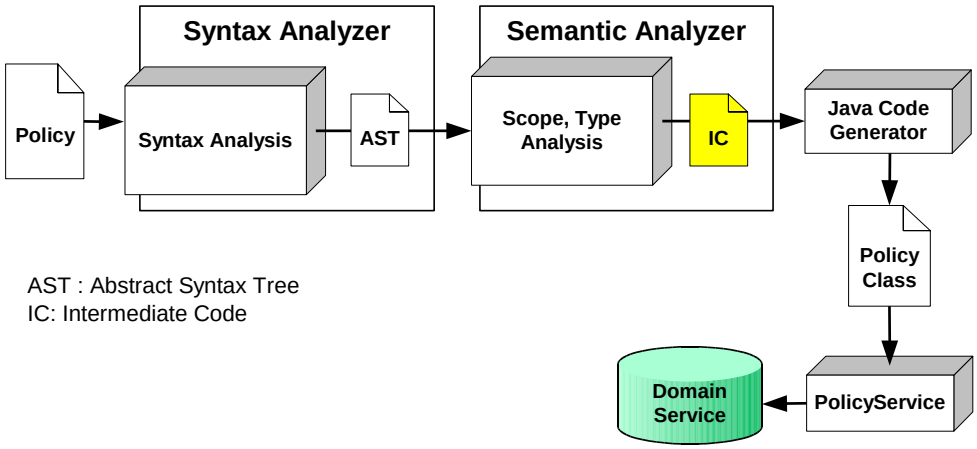

Figure 3.4 Policy compiler structure

The following is an example of an obligation policy type from the example scenario, and its generated Java code. Note that for each attribute type, the compiler builds an object that represents it. This object will then have to be serialized in some way to create the Java class (a static thing) and then reconstructed when the Java class is instantiated. In the following example it is assumed a method called “serialize” on

each of the object types to create a serialized representation of the object, and then a method reconstruct to re-create the object from that format. This can be simplified if we use the Serializable interface of Java. This will allow us to serialize and reconstruct an object using the build-in functionality of Java. Further investigation of this is required.

```
type mustdo spaceExceedT(actor backupAdmin) {
      target ms = /system/servers/mailServer;
      action ms.notifyQuota(uid);
      when userExceedQuota(uid);
}
// backup administrator (actor) sends a notification to a
user when that user's disk space quota is exceeded.
```

Figure 3.5 Example obligation policy

The Java code generator receives the intermediate code constructed by the main phases of the compiler for the obligationpolicy shown in **Figure 3.5**, and generates the Java class shown in Figure 3.6 as a subclass of the predefined Mustdo class (Figure 3.7). The mustdo class maintains the various policy elements and allows access to them. Since the objects for the various elements are already constructed, the Java code generator simply serializes them and then reconstructs them when the spaceExceedT class is instantiated, so that they can be set in the super class.

```
import java.io.*;
import java.util.LinkedList;
import policeToolkit.compiler.codeGen.policyObjects.*;
import policeToolkit.compiler.codeGen.policyObjects.events.*;
import policeToolkit.compiler.codeGen.policyObjects.constraints.*;

public class spaceExceedT extends Mustdo implements Serializable {
  public spaceExceedT(String name, ActorTarget backupAdmin)
  {
    // set the name and domain path of the policy
    super(name, "/managementInfo/pol/man");
    // any actual parameters to the instantiation are stored in a
    // linked list.
    LinkedList actualParams = new LinkedList();
    actualParams.add(backupAdmin);
    super.setActualParams(actualParams);

    // set the actor object
    super.setActor(backupAdmin);

    // target policy element : bytes for serialised target object
    byte[] targetBytes = { 0xffffffac, 0xffffffed, 0x0, 0x5, 0x73, ... };
    // reconstruct the target object
    ActorTarget target = null;
    try {
      ByteArrayInputStream in=newByteArrayInputStream(targetBytes);
      ObjectInputStream ois = new ObjectInputStream(in);
      target = (ActorTarget)ois.readObject();
    } catch (IOException e) { e.printStackTrace();
    } catch (ClassNotFoundException e){e.printStackTrace();}
    super.setTarget(target); // set the target object
```

```
    // action policy element : bytes for serialised action object
    byte[] actionBytes = { 0xffffffac, 0xffffffed, 0x0, 0x5, 0x73, ... };
    // reconstruct the action object
    MustdoAction action = null;
    try {
      ByteArrayInputStream in =
                          new ByteArrayInputStream(actionBytes);
      ObjectInputStream ois = new ObjectInputStream(in);
      action = (MustdoAction)ois.readObject();
    } catch (IOException e){ e.printStackTrace();
    } catch (ClassNotFoundException e){ e.printStackTrace(); }
    // set the action object
    super.setActions(action);

    // constraint policy element: bytes for serialised constraint object
    byte[] constraintBytes = { 0xffffffac, 0xffffffed, 0x0, 0x5, 0x73 ...
    };
    // reconstruct the constraint object
    Constraint constraint = null;
    try {
      ByteArrayInputStream in = new ByteArrayInputStream(constraintBytes);
      ObjectInputStream ois = new ObjectInputStream(in);
      constraint = (Constraint)ois.readObject();
    } catch (IOException e) { e.printStackTrace();
    } catch (ClassNotFoundException e) { e.printStackTrace(); }
    super.setConstraint(constraint); //set the constraint object
  }
}
```

Figure 3.6 Generated Java code snapshot

```
//The predefined classes are:

public class Mustdo extends BasicPolicy implements MustdoI {
  MustdoActionI mustdoActions;
  EventI event;
  public Mustdo (ActorTargetI s, ActorTargetI t, ConstraintI c)
  {
    super(s, t, c);
  }
  public void setMustdoAction(MustdoActionI oa)
  {
    mustdoActions = oa;
  }
  public void setEvent(EventI e)
  {
    event = e;
  }
  ... methods to implement the Interface go here
}

public class BasicPolicy implements BasicPolicyI {
  ActorTargetI actor;
  ActorTargetI target;
  ConstraintI constraint;
  public BasicPolicy (ActorTargetI s, ActorTargetI t, ConstraintI c)
  {
     actor = s;
     target = t;
     constraint = c;
  }
```

```
    ... methods to implement the Interface go here
}
```

Figure 3.7 Predefined classes

Police uses an integrated development environment (IDE) which is a modified version of the system developed by Ponder group for the specification of policies. The policy editor tool is integrated with our policy compiler and provides an easy to use development environment for specifying, reviewing and modifying policies. Templates can be used to create policies easily, and the domain browser can be invoked to select the subject and target domains for policies. Existing policies and policy types can be selected from the directory with the aid of the domain browser, loaded into the editor, modified, recompiled and stored back to the directory. Code generators added to the compiler framework, are accessible and can be enabled from within the editor to select the type of code to be generated.

SableCC is an object-oriented compiler framework developed at School of Computer Science McGill University, Montreal [37]. It is an object-oriented framework that generates compilers (and interpreters) in the Java programming language. This framework is based on two fundamental design decisions. Firstly, the framework uses object-oriented techniques to automatically build a strictly typed "abstract syntax tree" (AST) that matches the grammar of the compiled language and simplifies debugging. Secondly, the framework generates "tree-walker" classes using an extended version of the "visitor design" [45] pattern which enables the implementation of actions on the nodes of the AST using inheritance. These two design decisions lead to a tool that supports a shorter development cycle for constructing compilers. Written in Java language, the SableCC includes all the following features:

- Deterministic Finite Automaton (DFA) based lexers with full Unicode support and lexical states.
- Extended Backus-Naur Form grammar syntax. (Supports the *, ? and + operators).
- LALR(1) based parsers.
- Automatic generation of strictly-typed abstract syntax trees.

- Automatic generation of tree-walker classes.

As a new approach for compiler tools, the compiler place in the development cycle has been reduced to merely build an initial object-oriented framework that is based solely on the lexical and grammatical definition of the compiled language. This has the advantage of limiting framework modifications to the case where the grammar of the compiled language is changed.

In the generated framework:

- The parser automatically builds the AST of the compiled program.
- Each AST node is strictly typed, ensuring no corruption occurs in the tree.
- Each analysis is written in its own class. Writing a new analysis requires only extending some tree walker class and providing methods to do the work at appropriate nodes.
- Storage of analysis information is kept in the analysis class itself, outside the definition of node types. This ensures no modification to a node type is needed when a new analysis is added to or removed from the compiler.

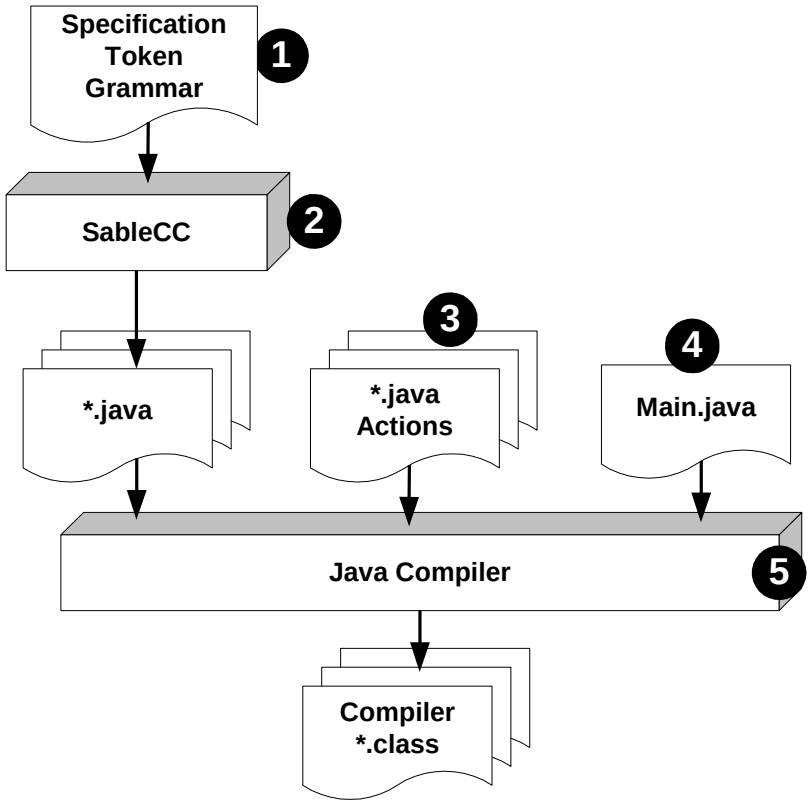

Figure 3.8 Steps to create a compiler using SableCC

The framework makes extensive use of object-oriented design patterns to achieve modularity of code. The resulting compiler becomes a very maintainable compiler.

Producing a compiler using *SableCC* requires the following steps (as shown in Figure 3.8):

1. Creating a SableCC specification file containing the lexical definitions and the grammar of the language to be compiled.
2. Launching SableCC on the specification file to generate a framework.
3. Creating one or more working classes, possibly inheriting from classes generated by SableCC.
4. Creating a main compiler class that activates lexer, parser and working classes.
5. Compiling the compiler with a Java compiler.

3.2.2.3 Domain Service and Policy Repository

The Domain Service maintains policies and other enforcement data such as information model of the managed system. Thus, LDAP is used for both storing policies and for grouping actor/target objects whereas the IETF framework uses directories only as policy repositories.

In large-scale systems it is not practical to specify policies for individual objects and so there is a need to be able to group objects to which a policy applies [46]. Domains provide a flexible means of grouping objects to which policies apply and can be used to partition the objects in a large system according to geographical boundaries, object type, responsibility and authority or for the convenience of human managers. The benefits of the domain-based approach are [46]:

- a policy applying to a domain will propagate to its sub-domains thus applying to large numbers of objects and providing scalability and,
- by moving an object from one domain to another its policies will be automatically replaced with those applying to the new domain, without the need to modify the policies or manually manage the association between policy and managed objects.

- when new objects are added or removed from the system they can simply be included or removed from relevant domains, without the need to modify the policies or manually manage the association between policy and managed objects.

Membership of a domain is not defined in terms of a predicate on object attributes. A domain does not encapsulate the objects it contains but merely holds references to object interfaces, and it is thus very similar in concept to a file system directory but may hold references to any type of object. A domain, which is a member of another domain, is called a subdomain of the parent domain. A subdomain is not a subset of the parent domain, in that an object included in a subdomain is not a direct member of the parent domain, but is an indirect member. Path names are used to identify domains, e.g., `/domain2/domain1`, where '/' is used as a delimiter for domain path names. Policies normally propagate to members of subdomains.

Domain scope expressions (DSE) introduced in [47] are used to combine domains to form a set of objects for applying a policy to (i.e. for the specification of actors/targets of policies). Domain scope expressions can be used to combine domains to form a set of objects for applying a policy, using union, intersection and difference operators, e.g., a scope expression `@/lan/routers + @/wan/routers` would apply to members of both domains and `@/hardware/routers ^ @/lan/routers` applies only to the direct and indirect members of the overlap between the two domains. In Police the domain scope expression is adopted as in the Ponder except several simplifications.

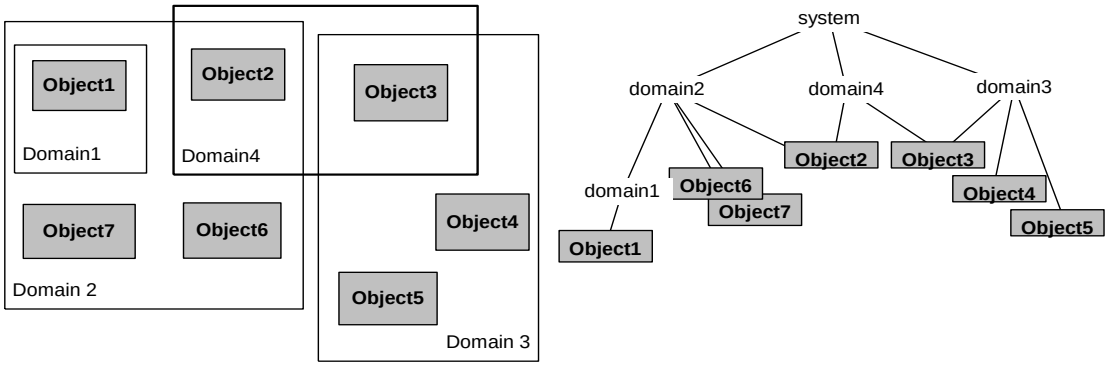

Figure 3.9 Sample domain structure

The Domain Service manages a distributed hierarchy of domains and supports efficient evaluation of actor and target sets at runtime or policy specification phase. A sample domain structure is shown in Figure 3.9. The policy enforcement mechanism must interact with target objects, both invoking operations and querying attributes. Thus, the Policy Management Server must have a priori knowledge of the managed objects in interface level, before the policies are specified. This is achieved with an information model stored in the Domain Service.

In the Domain Service, the managed objects are represented with classes, so-called Model Classes which define an element schema of the managed objects. Each model class must include the real management interfaces implemented by the real object without implementation details as shown in Figure 3.10.

```
Class SerialPort {
  int portCount;
  //...
  public openPort (int portID);
  public closePort (int portID) throw
  Exception;
  //...
}

//sample relationship
Relation between Object1.method1 and
Object2.method3;
```

Figure 3.10 Model class for a serial port and a relationship

The information stored in the Domain Service relevant to the entire management process is defined under the term of System Information Model (SIM). This is composed of a meta information model including management components (PC) and policies, as well as a management schema including managed objects which participate in the management process and relationships between them. The relationships between managed objects are the essential elements of Police conflict handling method. Figure 3.11 shows a diagram for the management schema.

In the meta model, everything is defined as a managed object. This includes domains, policies and managed objects. A domain may include any number of managed objects, for which the domain is the parent. Policies are defined over sets of objects formed by applying set operations, such as union, intersection and difference

to the objects within domains. Actors and targets of policies are defined in terms of domains, and this is indicated with the dependency line in the figure. Enforcement components are responsible for the enforcement of policies in the runtime management system. Policy management components are automated components responsible for enforcing policies. Basic policies are distributed to the enforcement components, hence the usage path line between the enforcement component class and the basic policy class.

In the management schema, the managed objects are represented with Model Classes as explained above. In addition, the application specific relationships are described between the managed objects and their methods. A node discovery mechanism and administrator intervention capability (for the nodes not supporting auto discovery) is employed to keep the element schemata up-to-date.

As mentioned before, the SIM models the information of the system to be managed with policies. For referencing purposes, the model classes to be put in the Domain Service will fulfill the requirements of Common Information Model of DMTF.

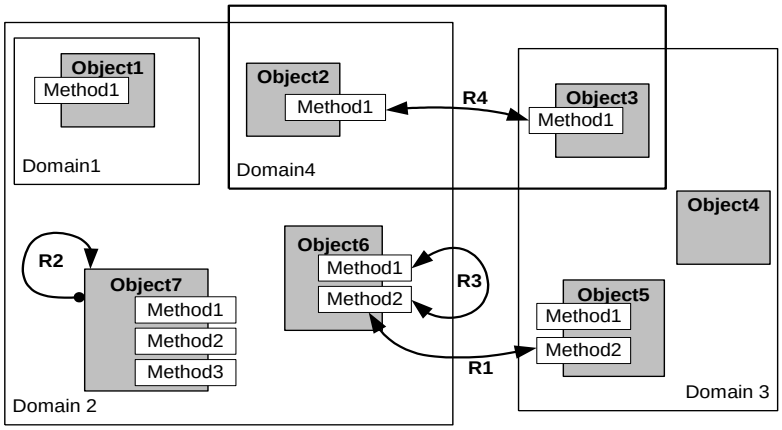

Figure 3.11 A graphical representation of sample management schema

3.2.2.4 Policy Deployment Service

The distribution of policies is initiated by the policy management service and performed by Policy Deployment Service. Policy Objects (POs) are generated and maintained by the policy service to manage policies at runtime. This includes the distribution of the policies to their enforcement components, and the handling of dynamic domain-membership changes to the objects to which the policies apply. The fact that policies explicitly define their actors and targets makes the automated distribution of policies possible. All kind of policies are distributed to their actors.

3.2.2.5 Police Policy Enforcement Point (PPEP)

The PPEP is the enforcement point of our model and a Policy Management Server can handle more than one PPEP. PPEP design and implementation is subject to my thesis and PPEP is explained in detail in next sections.

3.2.3 Policy Distribution

Both cando and mustdo policies are distributed in the same uniform way to the PCs near the actors that it makes our architecture simpler.

Policy Management Service generates a policy enforcement object for each policy that Policy Deployment Service distributes to the related enforcement platforms. The PPEPs analyze the deployment objects they receive and enlarge their behavior according to the content of them.

Domain Server stores the relations between the PPEPs and managed devices. Each managed device has one PPEP and each PPEP controls only one device. Managed device include multiple managed nodes in it. This information can be entered to Domain Server manually or PPEP registers itself to Domain Server on startup. On registering process PPEP send the managed nodes it has the control and the events that can be created by the managed device.

When Deployment Service is about to send an enforcement object to a managed device, firstly it learns the PPEP that is responsible for that device from the Domain Server. PPEP is searched from the Domain Server according to the “actor” of the policy.

3.3 Conflict Handling

Policy conflict occurs when the actions of two rules (that are both satisfied simultaneously) contradict each other. The entity implementing the policy would not be able to determine which action to perform. The implementers of policy systems must provide conflict detection and avoidance or resolution mechanisms to prevent this situation. "Policy conflict" is contrasted with "policy error" [3].

Policy errors occur when attempts to enforce policy actions fail, whether due to temporary state or permanent mismatch between the policy actions and the device enforcement capabilities [3].

3.3.1 Conflict Classes

Main conflict classes [38] are modality conflicts and objective conflicts.

Modality Conflicts [38,39] may arise if there are both positive and negative authorization or obligation policies with same actors, targets and actions. Modality conflicts can be detected through static, syntactic analysis of the policy specifications. Most existing work in detecting conflicts relates to modality conflicts. Due to nonexistence of negative policies, Police does not suffer from modality conflicts.

Application-Specific Conflicts arise from the semantics of the policies. They cannot be determined directly from the policy specifications without additional information specifying what conflict is. In the literature, the Metapolicy [38,39] concept is used to represent this additional information, which is a constraint about permitted policies (policies about policies).

In [38], it is claimed that if there is not common object of two policies (disjoint policies) there is no possibility of conflict too. But, we think that this assumption is not true for some application-specific conflicts. There may be implicit relationships between objects and consequently between policies applied to them. This may cause application-specific conflicts implicitly.

One example for application specific conflicts can be expressed as "*There should be no policy pair authorizing an actor to perform two separate operations that must be performed by different actors*" [39]. If one policy says "accountants can sign a cheque" and another policy says "accountants can write a cheque" but there may be another policy saying "same accountant cannot write and sign a cheque". In this case the first policies allow an accountant to write and sign a cheque but these two conflict the last one.

Metapolicy-based approach cannot always fulfill the requirements to solve the problem of application-specific conflicts. Since, the administrator must specify metapolicies for all possible runtime situations, which may result in conflicts. It is almost impossible for the administrators to overlook all implicit relationships may cause conflicts in the system being managed. Moreover, in many cases, the fact that there are policies related to each other does not mean a conflict will occur at runtime.

In contrast to working groups using meta-policy concept to solve application-specific conflicts, we have developed a novel policy conflict detection mechanism based on the relationships between the objects of the system to be managed. On considering these associations among actors or targets, the relationships between the policies to be enforced on these objects can also be determined. Because, object relationships implicitly show the relationships of policies applied to them. The Policy Controllers investigate the likelihood of conflicts at runtime by analyzing the relations between policies. Then, they enforce their actors according to this analysis.

3.3.2 Police Conflict Handling Method

Due to nonexistence of negative policy concept, Police language does not suffer from modality conflicts. In case of application-specific conflicts, in contrast to working groups using meta-policy concept, a different policy conflict detection mechanism is developed [33]. This method is based on the relationships between the objects of the system being managed.

In order to make Police Conflict Detection model run, all associations between managed objects must already be modeled in Domain Service before specifying policies. By evaluating these associations, direct or indirect relationships between the policies to be enforced on these objects can also be determined. The Policy Controllers (PCs), enforcement agents, investigate the likelihood of conflicts at runtime by analyzing these relations between policies. Then, they enforce their actors according to the result of these investigations. To do this, the PCs must first look for any relationship interesting the managed objects referenced in the policies for which these PCs are responsible. If there is an association, a PC has to get and run the Evaluation Code related to that association from the repository. On running the Evaluation Codes, the real runtime situations about the conflicts can be obtained. Unlike Ponder performing a static conflict analysis on policies at Policy Server, postponing the decision of whether there is any conflict, until enforcement time makes Police model more consistent and reliable. In addition, using Evaluation Code concept, instead of meta-policies provides more flexibility.

Thus, the policy conflict analysis is performed in a distributed manner. The administrators specifying policies do not have to worry about possible conflicts. The policy conflict analysis logic is embedded into the managed system itself.

4 ENFORCEMENT MECHANISM OF DIFFERENT SYSTEMS

Policy enforcement is the execution of a policy decision and Policy Enforcement Point is a logical entity running on or within a resource (e.g., device) that enforces policy decisions and/or makes a configuration change.

In this chapter following Policy Based Management systems are studied: Active PBM [17], PONDER [48], Script MIB, HP OpenView PolicyXpert and IPHighway Open Policy System. For each framework, general architecture and the Policy Enforcement Point architecture is given.

4.1 Active PBM

Active PBM [17] presents a solution using Active Network Technology to facilitate the deployment of PBM. This approach considers the network as an active part and a manageable resource of the running services and not only a passive communication tube and intends to use capsules (active packets) as a flexible way to exchange policies via the network. This section aims to describe Active PBM which is mostly related with SLA (Service Level Agreement) and QoS and policies related with them.

Active networking emerged from discussions within the Defense Advanced Research Projects Agency (DARPA) research community. The aim of the Active Networking today effort is to design, develop and implement new communication architectures that allow rapid, safe and dependable creation, reconfiguration, and deployment of advanced networking services, protocols and management components.

The Active Networking paradigm achieves this goal on one hand through the concepts of self-directing data units called Capsule, and Open Programmable Nodes. Active packets can direct their own processing and deliver new services to the interior network nodes.

Programmable Nodes permit remote and secure processing and building of enhanced network services from generic network software elements. The conjunction of these concepts will ultimately result into networks with greatly improved communication services that meet user requirements efficiently and securely [49]. This is motivated by the desire to solve some of the problems encountered in large scale distributed and real-time systems such as the volume and complexity of the tasks, latency, delays, and others.

The majority of current communication system architectures employ the client-server method that requires multiple transactions before a given task can be accomplished. This can lead to increase signaling traffic throughout the network. This problem can rapidly escalate in an open network environment that spans multiple domains. As an alternative solution, active networking can shift the computations or interactions to the remote host by moving the execution there. The main motivation of the use of active networking technology in this work is driven by the desire to automate the control and management processes by allowing for more programmability of the network to rapidly customize the provision of new information and telecommunication services [50]. Hence, capsules can also be used to implement Service Level Agreements (SLAs, a formal negotiated agreement between two parties) [51] between different actors of the network and service era. Capsules can then be used as brokers or mediators between end users and service providers in order to implement the SLA. In this way, complex QoS metrics specifications (from end user's point of view) can be transferred in a simplified manner. ISP nodes can therefore negotiate with users' node in order to meet the required service [52].

4.1.1 Active PBM Architecture

The system architecture shown Figure 4.1 comprises two main components, the PEP and the PDP. These two environments differ mainly in term of functionality's and localization. In fact the PEP environment is located at the router boundary while the PDP environment is a standalone system. The PDP (Policy Decision) can enforce policies in the network by sending capsules to PEP (Policy Enforcement Point) enabled IP routers.

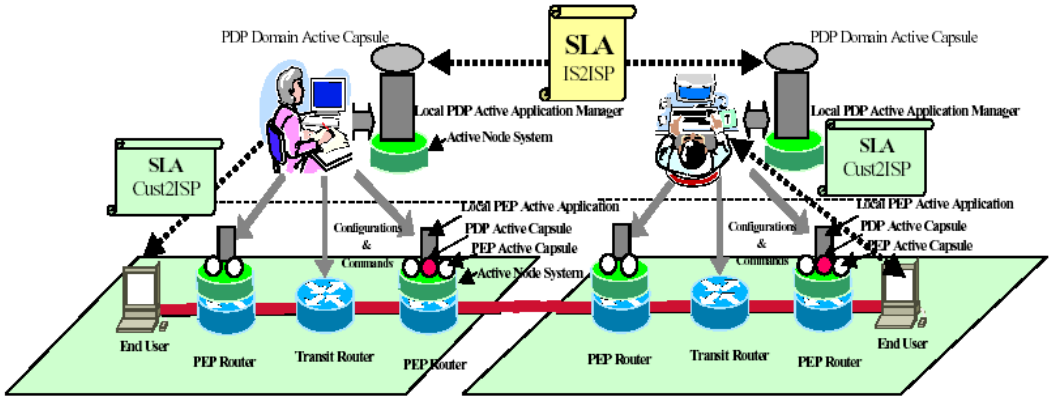

Figure 4.1 Architecture of Inter-domain A-PBM

The open architecture defines a set of active services and capsules depending on their respective roles in the architecture:

- Local PEP Active Service: It is an Active Service that performs local routine control/management PEP functions. It performs mainly metering and enforcement functions as well as the sending of PEP capsule when it needs to interact with the Local PDP active services manager for decisions.
- PEP Capsules: These are used mainly as autonomous negotiator capsules between PEP and PDP within the same domain. The *PEP Capsules* are used to obtain decisions from PDP. The PEP *Capsule* carries all the information regarding the ongoing connection. It is sent by the PEP to the PDP in order to notify a particular event in the network (RSVP opening request [53], QoS degradation, etc).
- Local PDP Active Service Manager: It is an Active Service that takes decision regarding the information that is carried out by the PEP Capsules. It interacts with the various databases (policy rules DataBase , MIB, security DB, etc) in order to retrieve the rules that can be triggered. Once the decisions are identified, it sends this by the PEP Capsules. If any configuration related to new policies is defined, it sends PDP Capsules, to remote PEP Active Services to perform the new configuration. It takes also into account inter-domain interactions, when a decision needs to be negotiated with remote domain PDP Active Services Manager. When interacting with remote domain, it sends PDP Inter-Domain Capsules.

- PDP capsules: When a PDP active service has taken a decision it sends a PDP capsule to enforce policies directly in the PEP active services component in all the network elements that are concerned by this new decision.
- PDP Inter-Domain capsules: When a PDP has to take a decision related to an inter-domain connection, it has to identify the set of remote domain need for the connection and send an Inter-Domain capsule to negotiate the term of services needed by the customer.

4.1.1.1 Operation of Active PBM

In the case of two ISP domains interconnecting the customers premise networks, the deployment of the different capsule in the global distributed architecture is described below.

The local PDP active services manager is responsible for collecting information related to the entire domain. If any change occurs in the network such as an RSVP connection request, the local PEP active services on the ingress router sends a PEP capsule to the PDP active services manager. The sent capsule contains all the information needed to identify the source of the request (customer) and the destination of the call (calling party) as well the parameters related to this event (for example QoS parameters for the request RSVP connection). Based on this information, the local PDP active services manager retrieve related information and policies from the policy DB, the MIB and the security server using different types of protocols such as LDAP, SNMP or any other protocol that permit to retrieve information from a database. Then, the local PDP active services manager tries to trigger any policy rule that can be triggered regarding the information carried by the PEP capsule. If the connection doesn't span a different ISP domain, the PEP capsule is sent back with the response to the local PEP active services. If the decision needs to interact with remote ISP, the local PDP active services manager sends a PDP inter-domain capsule to remote ISPs with all information related to the requested service as well as information permitting to identify the initiating domain. The remote local PDP active services manager gets the necessary information from the remote (received) PDP inter-domain capsule. According to this information, it tries to trigger any policy rule that defines the ISP-to-ISP policy rules between this ISP and the initiating ISP defined within the SLA. If the service is accepted the PDP inter-

domain capsule is sent back to its domain with all the information related to the decision. The local PDP active services manager retrieves the information and takes a final decision regarding the request service.

When the final decision is taken, local PDP active services manager of each domain that intervened in the final decision has to configure its own equipment in order to enable the customer service to be operational. This means for instance to enforce policy directly into equipment using PDP capsule. Consequently PDP capsule will move from equipment to another in order to enforce locally the policy by interacting with the local PEP active services.

4.1.2 Policy Enforcement Point architecture

Policy carried by capsules will force the PEP to enforce local differentiation regarding the inbound user packets. Thus it is possible to differentiate between user packets and adapt the provided QoS depending on the ISP policies and the SLA agreed with the customers.

Active PBM PEP receives a Policy Capsule in two ways:

- PEP sends a request to PDP and PDP sends the Policy Capsule back after a query in the DataBases or an Inter Domain communication. This way uses outsourcing policy execution approach.
- PDP makes a decision and sends the Policy Capsule directly to PEP. This is provisioning execution approach.

When the service requested by a customer span a number of ISPs, a negotiation process is launched between the ISPs. The goal of this process is to find out the best choices (route, price, etc) to ensure that the QoS requested by the customer in the SLA will be fullfiled.

The PEP environment is designed light in the sense that it should not require a lot processing and memory resources and should be as fast as possible.

The execution environment at the PEP point is an active node with agency services. The architecture is shown in Figure 4.2. The agency services are DMIB objects located in the router. In this scheme, an active node ANTS [54] based on a Java Virtual Machine (JVM) is proposed as a middleware solution. Based on the PDP decision, the local PEP services assign a policy to users' connection. In order to have

a standard interface between the capsules deployed on the router and the embedded hardware and software, an Object Request Broker (ORB) is used between the JVM and the Kernel. The reason for using an ORB is to provide in one hand all the support for services management and mobility and on the other hand a standard L interface [55] to interact with router kernel, since there is a wide variety of hardware and software within routers from different vendors.

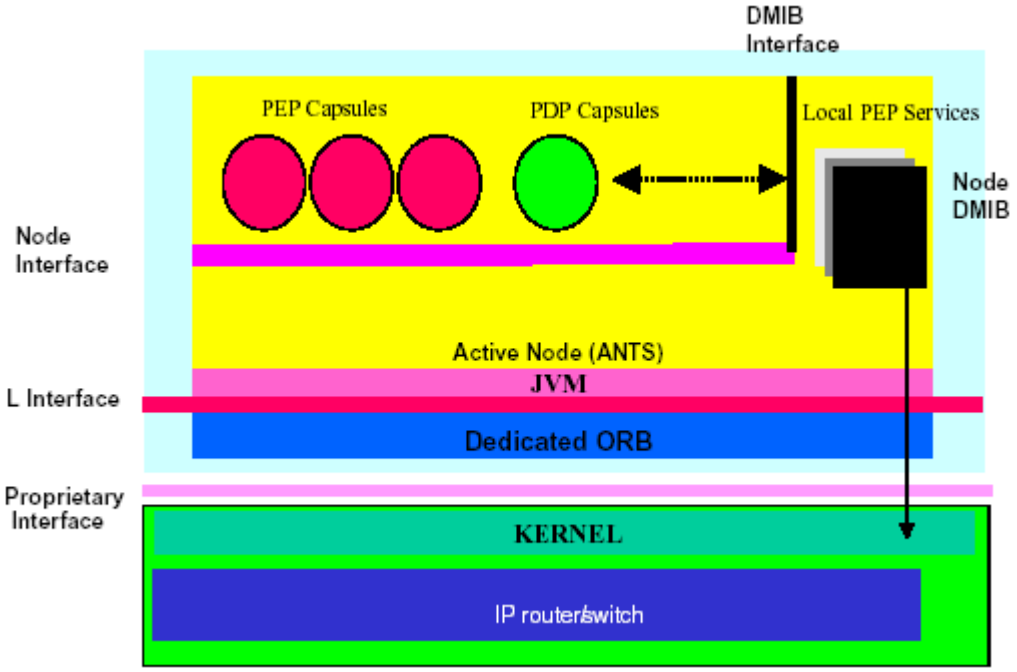

Figure 4.2 PEP environment architecture

4.2 PONDER

PONDER deployment model is developed by Imperial College and is object-oriented and addresses the instantiation, distribution and enabling of policies as well as the disabling, unloading and deletion of policies. It forms part of the run-time support for Ponder language for specifying both management and security policies. Policies apply to domains of objects, and a policy applying to a domain will propagate to all objects of that domain including objects of nested subdomains. Both domains and policies are mapped to objects [48,56].

4.2.1 PONDER System Architecture

Ponder has four basic policy types: authorizations, obligations, refrains and delegations and three composite policy types: roles, relationships and management structures that are used to compose policies; domains for hierarchically grouping

managed objects, events for triggering obligation policies, and constraints for controlling the enforcement of policies at runtime.

Figure 4.3 illustrates the architecture of the management system. It includes three supporting services: a domain service, a policy service and an event service. The "policy service" acts as the interface to policy management; it stores compiled policy classes, creates and distributes new policy objects. The "domain service" manages a distributed hierarchy of domain objects and supports the efficient evaluation of subject and target sets at run-time. Each domain object holds references to its managed objects but also references to the policy objects that currently apply to the domain. In concept, the domain service is similar to a directory service such as LDAP, with extensions to allow changes to the membership of a directory to be distributed to interested parties, e.g. via events. The domain service can also be implemented by database systems. The "event service" collects and composes system events as well as those from the managed objects in the system, and forwards them to registered policy management components to trigger obligation policies. The event service exposes a publish/subscribe interface whereby clients can subscribe to receive certain types of events.

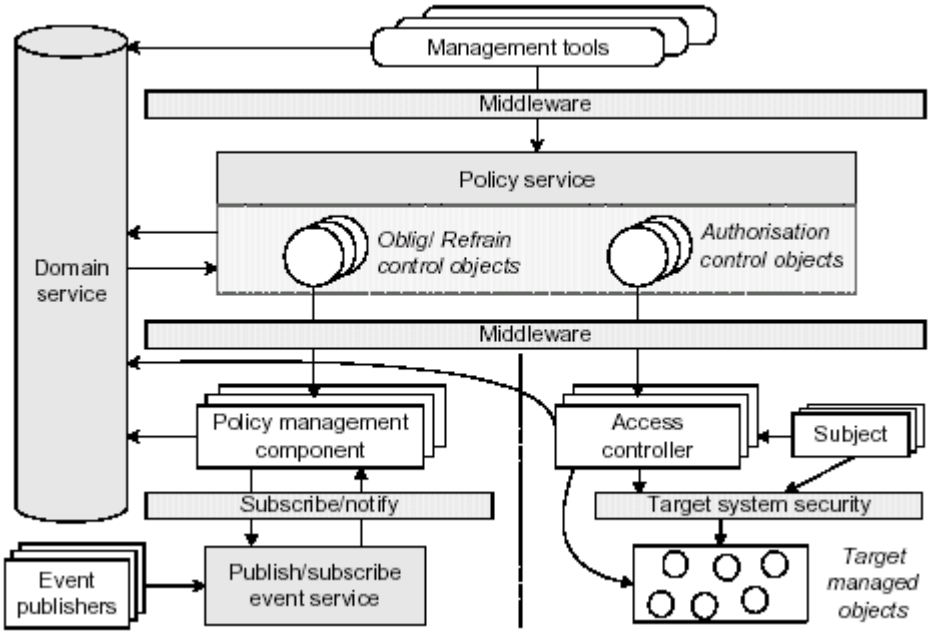

Figure 4.3 PONDER Management system architecture

4.2.2 PONDER Enforcement Architecture

Enforcement agents are objects that implement a policy enforcement interface. This interface supports the loading, enabling and disabling of enforcement objects disseminated from policy objects. In PONDER enforcement objects are created by policy objects and moved to enforcement agents when the policy loading is requested. Enforcement agents store the passed enforcement objects in a local policy table and forward most policy operations to corresponding methods implemented by the enforcement object. Compiled enforcement classes provide the implementation code for enforcing policies and can be target-dependent and subject dependent. Thus a policy may have different implementations of the policy enforcement class. In order to handle dynamic changes in domain object membership, the policy enforcement interface also has methods for the addition and removal of objects.

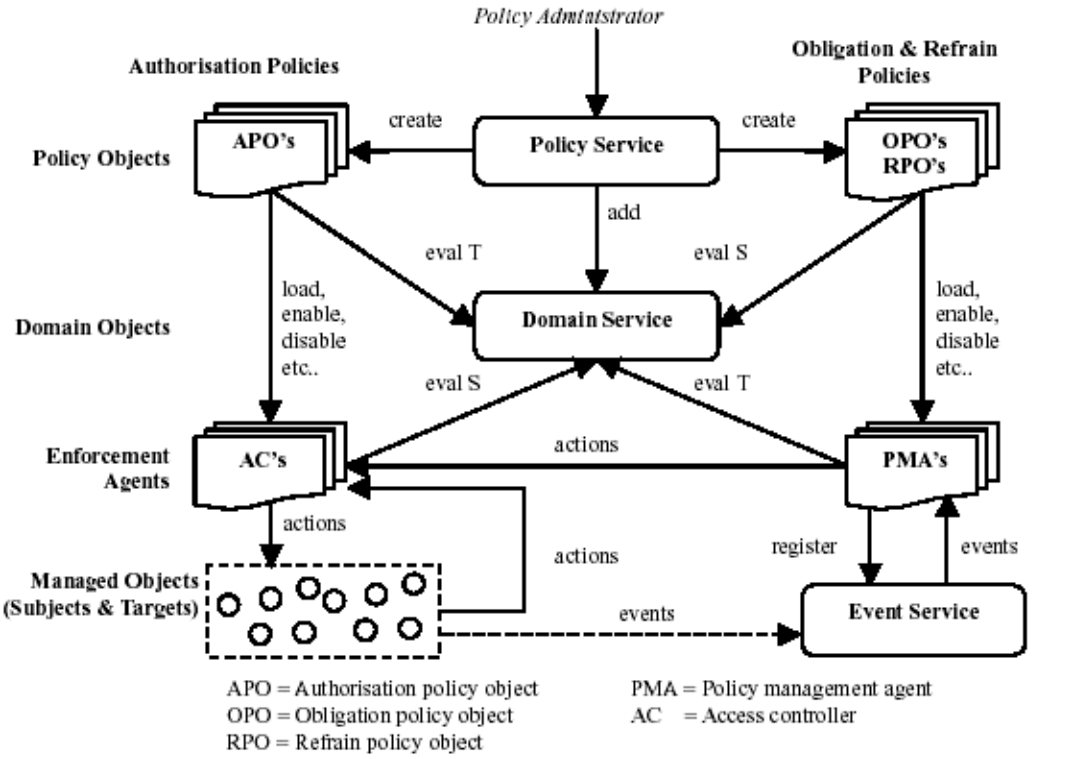

Figure 4.4 Overview of policy deployment model

As shown in Figure 4.4 for obligation policies, Policy Management Agents (PMAs) register with an event service to receive relevant events generated from the managed objects of the system. On receiving an event, the PMA queries the domain service to determine the target objects used in the obligation method and performs the policy actions, provided no constraint or refrain policy prevents the action. When a

managed object is about to perform an action on another object, first it gets access permission from access controller.

The Event Service collects and composes events from the underlying systems and from the managed objects in the system, and forwards them to registered policy management agents triggering obligation policies. After a policy object is instantiated, it can be loaded into its enforcement agents, and once loaded, it can be enabled causing its enforcement agents to actively implement it. An enabled policy can be disabled and later re-enabled, or disabled and then unloaded, removing it from its enforcement agents. Unloaded (i.e. dormant) policies can either be reloaded or deleted. Figure 4.5 shows all the states for a policy object.

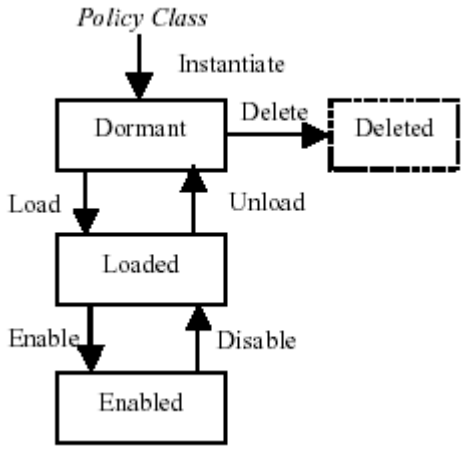

Figure 4.5 Policy object states

4.2.2.1 Policy Management Agents (for Obligation and Refrain Policies)

Policy management agents enforce all the enabled refrain and obligation methods for a subject. An overview of the operation of a policy management agent (PMA) is shown in Figure 4.6. The enforcement objects for obligation policies and refrain policies (Obligation Enforcement Objects (OEOs) & Refrain Enforcement Objects (REOs)) are loaded from corresponding policy objects (Obligation Policy Objects (OPOs) & Refrain Policy Objects (RPOs)) and stored locally (1). When an obligation policy is enabled its obligation enforcement object registers the obligation event specification along with a reference to an event handler with the event service (2). The event service processes events (3) and disseminates them to handlers based on their event specifications (4). On receiving an event, handlers check both the constraints of the obligation policy and all enabled REOs within the agent to check if

any REO disallows actions within the obligation method (5 & 6). If constraints and refrains allow, the event handler then calls the obligation method, which performs actions on managed objects (7, 8). Two interactions are omitted from Figure 4.6. Firstly, the event handler in the PMA queries the domain service in order to evaluate the target set on which actions are to be invoked i.e., the event handler effectively coordinates the execution of the obligation policy. Secondly, obligation policies are allowed to invoke actions internal to the PMA.

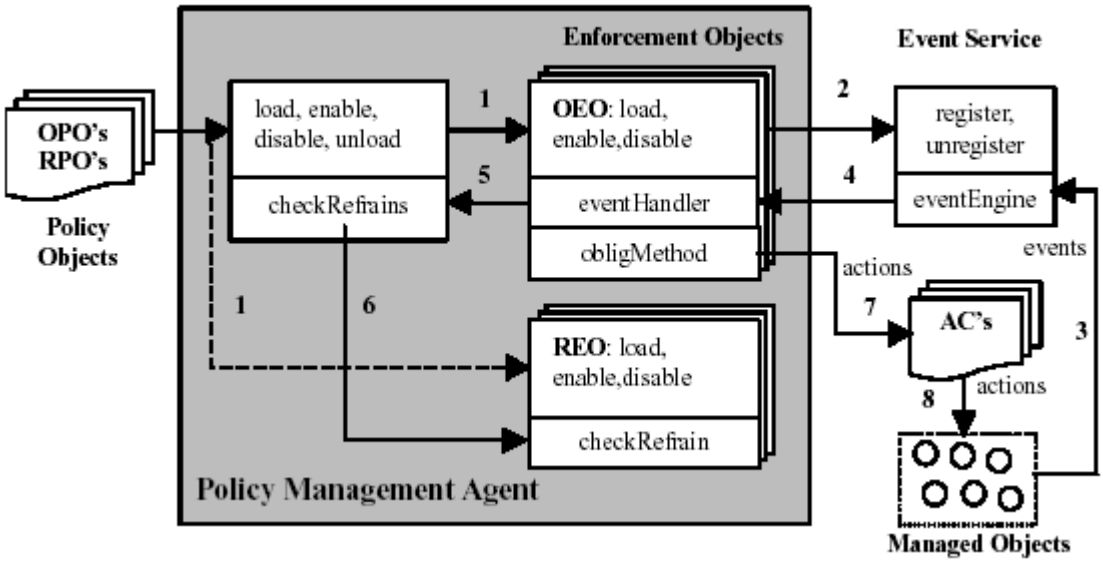

Figure 4.6 Overview of a Policy Management Agent

4.2.2.2 Access Controllers (for Authorization Policies)

Access controllers enforce all the authorization policies for one or more target objects. Access controllers are normally co-located with the targets that they protect. Unlike policy management agents, which can be generic, access controllers require close interaction with the underlying access control mechanism, for example, with the host operating system, or a firewall, or the method dispatch mechanism of a programming language. The means used to interact with each mechanism will vary. Access controllers follow the general approach for all enforcement agents. They implement the policy enforcement interface and provide methods to load, enable, disable and unload authorization enforcement objects (AEOs) similarly to PMAs and OEOs, except that AEOs are not event-driven. Authorization enforcement objects also provide methods for evaluating constraints and handling authorization filters.

When an action is “intercepted” by an access controller, it calls a “checkAccess” method to check whether the access should be permitted. This method will check, for example, that the subject of the action is in the subject set of the policy, that the target of the action is in the target set of the policy and that the action is a valid action for the target. The method will also evaluate all policy constraints and enforce any global rules for the systems, for example that access to the target object must only allowed if there is a positive authorization that allows the subject to perform the action on the target object, and no negative authorization that forbids the subject from performing the action.

4.3 Script MIB

The IETF has developed several specifications for policy-based configuration management. In addition, an infrastructure for distributed management by delegation has been specified. Script MIB Policy Based Management framework combines key concepts developed in these two efforts and proposes a policy-based configuration management architecture built upon the distributed management infrastructure. Two prototype implementations for managing differentiated services are discussed and evaluated [24].

The first solution is closely related to the SNMPconf approach. Policies are defined as programs. These are downloaded as ’scripts’ via the Script MIB and then executed by a common Script MIB runtime engine. The runtime acts as a PDP and sends configuration information to a local or remote PEP.

The second solution builds upon the Policy Core Information Model (PCIM). Policies are not defined as programs, but as groups of PCIM objects. Such policies stored in a repository are downloaded using the Script MIB directly to managed nodes. There, an interpreter for this special kind of ’scripts’ combines PDP and PEP functionality [24].

4.3.1 Script MIB System Architecture

The general architecture of the Script MIB based policy system is shown in **Figure 4.7**. This approach is based on the Jasmin Java runtime engine. Hence, policies are Java programs edited and compiled on a manager host. The created jar-Files are served to the Jasmin agents by a policy repository via HTTP. When the agent

activates a policy, e.g. on demand of a manager, it forks a Jasmin Java runtime engine and forces it via SMX to start a given Java "script" from the local script storage. To control and monitor the execution of policies, the management application communicates with the agent(s) via SNMP.

The policy specific components lie within the Java runtime engine, which makes use of Java class packages that support policies. A general policy package contains a number of abstract classes that represent policies (with their rules, conditions, and actions), events, and elements. Besides these abstract classes, domain-specific packages contain derived element and event classes, e.g. to represent classifiers, meters, markers, etc. in case of a DiffServ specific policy package. These domain-specific classes along with classes from the generic policy package, like time event classes, are used by policy "scripts".

The interaction between the domain-specific classes and the managed devices is based on interface drivers hidden from the package API. This allows supporting different interfaces, e.g., SNMP, COPS-PR, local APIs or CLIs, without the need to adopt policy scripts to these interfaces or protocols [57].

The policy manager must be aware of the roles associated to each network element, since it performs a policy selection in order to know which policies each agent must retrieve. The concept of roles is used to select the policies which apply to a certain network element and therefore need to be downloaded to the Script MIB agent. The policy management application needs to keep track of the roles of the network elements controlled by the Script MIB agent, and what policies apply for what role.

The Script MIB agent acts as PDP, retrieving policies from an HTTP or File Transfer Protocol (FTP) server that is acting as a policy repository. The runtime engine of the Script MIB agent is a policy engine that evaluates the policies delegated to it as 'scripts'. A policy evaluation may require monitoring network elements and executing actions by sending configuration messages to the network elements.

As shown in **Figure 4.8**, there are two manager-agent relationships, one between policy manager and PDP and one between PDP and PEP. For communication between policy manager and PDP, SNMP and the Script MIB are used. For communication between PDP and PEP several alternatives can be used: SNMP, COPS-PR, SSH/CLI, or a local interface, if PDP and PEP are co-located.

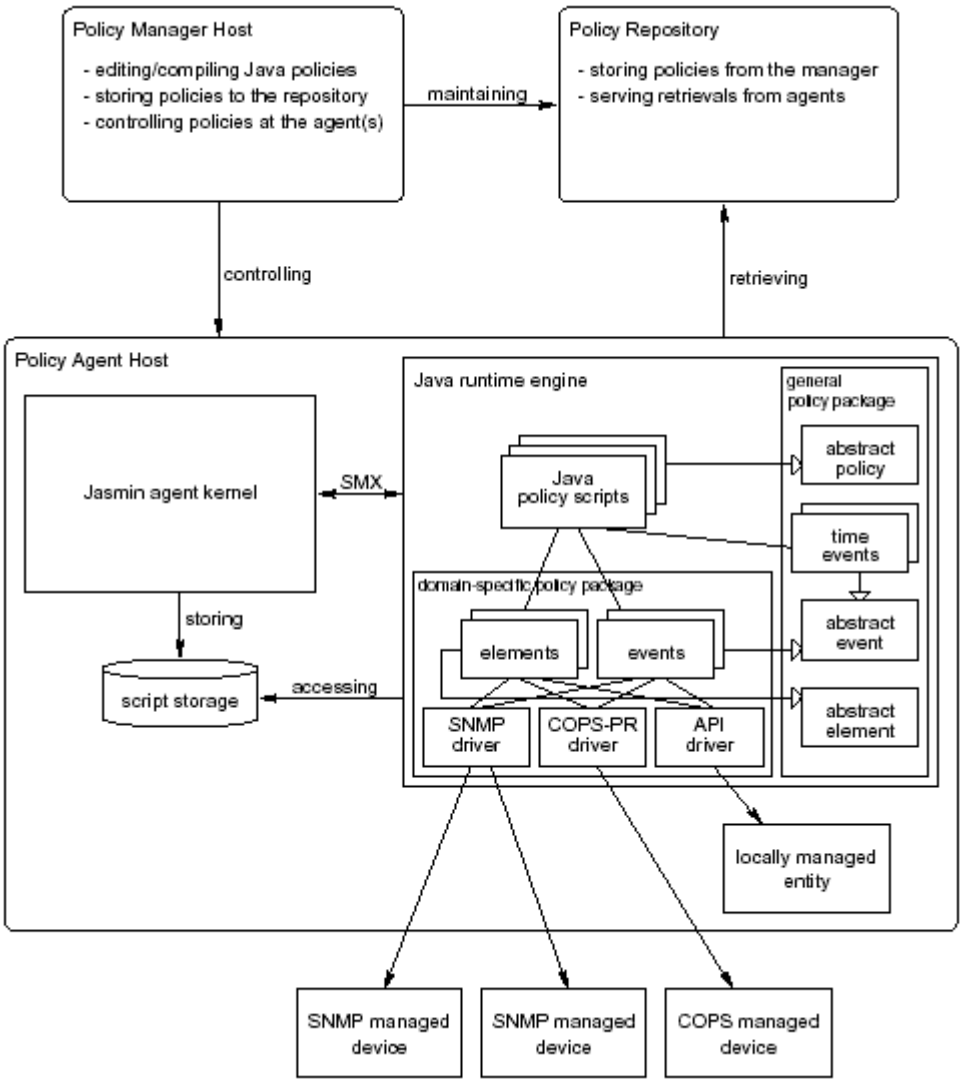

Figure 4.7 Architecture of PBM based on the JASMIN Java runtime engine.

While there is only one instance of policy manager and policy repository, there may be multiple PDPs and for each PDP there may be multiple PEPs. The architecture covers three levels of PDP distribution shown in **Figure 4.9**: (a) centralized with just a single PDP, (b) weakly distributed with several PDPs, but much less than the number of PEPs, and (c) strongly distributed with one PDP per PEP.

For centralized policy management there is no specific advantage in using the Script MIB compared to other policy management systems. Weakly distributed policy management is more scalable. The central PDP in (a) may become a bottleneck, if the number of policies and/or the number of PEPs increase too much. In (b) the bottleneck is removed by distributing the load to as many PDPs as required.

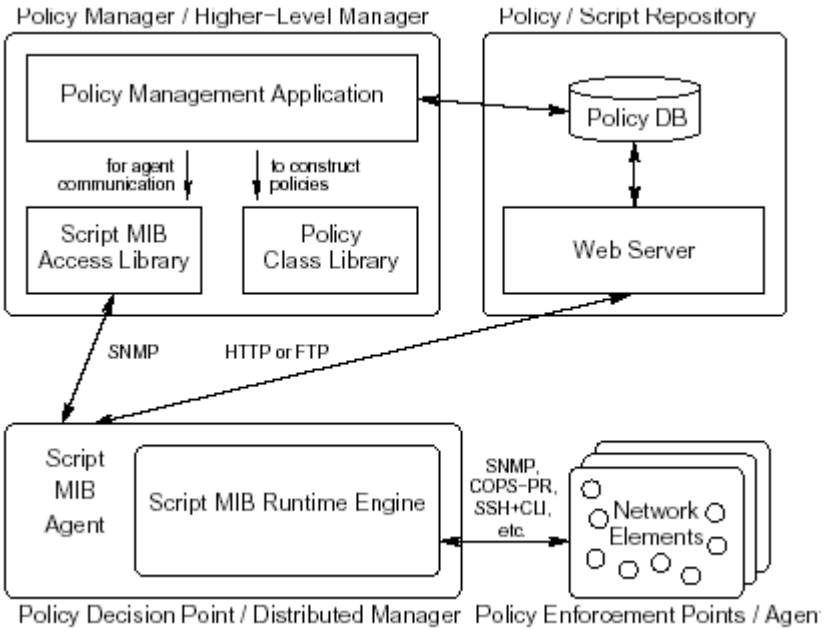

Figure 4.8 Architecture of the Script MIB based configuration management.

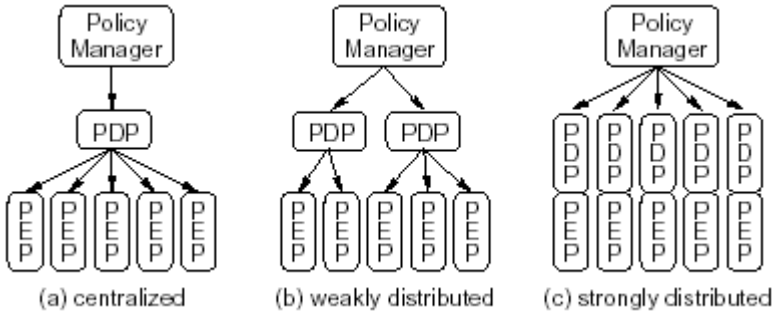

Figure 4.9 Different levels of PDP distribution.

In case of strongly distributed policy management, all network elements hosting a PEP also host a PDP. Here, no standardized communication between PDP and PEP is required anymore, because local proprietary communication may be used. The scalability is still higher than in (a) because policy information is considered to be more condense than the configuration information derived from it and because the policy manager now can select which policy to send to which network element. For example, in a DiffServ network management system with all routers hosting PDPs and PEPs, core routers would only receive and evaluate core router policies, while edge routers receive different and potentially more complex policies.

4.3.2 Enforcement (Policy Engine)

There are two approaches for the policy engine in Script MIB (**Figure 4.10**).

The first approach represents policies by program code. This matches the typical use of the Script MIB. A policy or a group of policies are represented by a program that is passed as a "script" to a Script MIB agent. At the agent the program is executed by a runtime engine for the used programming language. This runtime engine must provide a way of accessing the network elements to be configured by the policies, e.g. by offering a specific library.

The second approach represents policies by objects. A policy or a group of policies are represented by a set of objects. Again, the set is passed as a "script" to a Script MIB agent. There, the objects are evaluated by a specific policy runtime engine. The objects representing policies conform to PCIM, they contain data only and no code. Also, they are specific to an application domain, e.g. Policy QoS Information Model (QPIM) [58] for QoS. The policy runtime engine must contain implementations of all PCIM policy classes that are to be evaluated, and it must have access to the network elements to be configured by these policies.

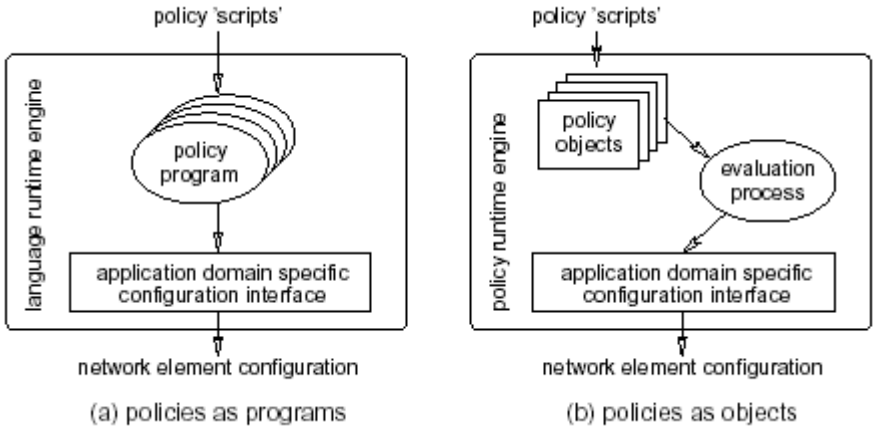

Figure 4.10 Two different approaches to PBM with the Script MIB.

4.3.2.1 Policies as Program Codes

This approach, shown in **Figure 4.11**, is based on an existing Script MIB runtime engine. Policies are represented as scripts written in a language supported by that runtime engine. A general policy management language extension provides interfaces for deriving and implementing policies, rules, conditions, actions, network elements, event generators and events. Domain specific language extensions provide abstract interfaces to network elements of a specific policy application domain. They allow policy scripts to retrieve element attributes and event notifications and to

correlate them to make policy decisions, so that they can in turn be used to configure network elements. Drivers realize the mapping between the domain specific interfaces and the underlying device-level mechanism to actually configure the network elements.

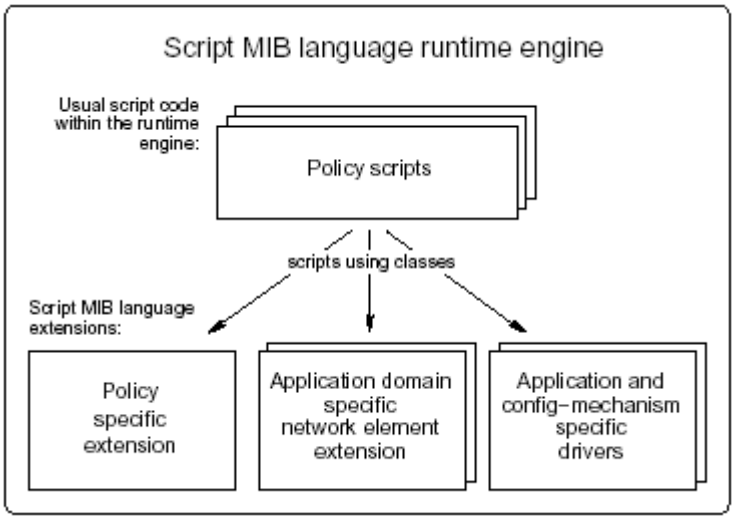

Figure 4.11 The standard Script MIB runtime engine

4.3.2.2 Policies as Objects

In the second approach, policies are coded not as programs running independently of each other, but as objects (or sets of objects) that are executed by a single policy evaluation process, see **Figure 4.12**. Therefore, the runtime engine cannot be anymore an interpreter of a common programming language. A special runtime engine for policy objects is required. In order to use existing standards as much as possible, PCIM and standards derived from PCIM is chosen as information model for the policy objects. Implementations of the classes these objects are instances of, must be available at the runtime engine. The policy evaluation process executes policy actions by using application domain specific interfaces. The functions that the policy runtime engine must offer are:

- Monitor domain specific attributes using domain specific objects that represent the configuration state of the network elements controlled by the Script MIB agent. These objects might use driver functions to provide an abstract interface to configure the underlying specific domain implementation.

- Receive orders from the Script MIB agent to add, remove, enable or disable policies.
- Handle the triggering of policies. The event that triggers the evaluation of a policy is encoded within the delegated policy object.
- Identify the set of target domain objects to which the triggered policy applies. The role of the policy is specified within the policy object.
- Compare the values of selected domain object attributes with the conditions specified within the triggered policy object.
- Execute the actions of a policy rule after the corresponding condition was evaluated to true. In general these actions are performed by modifying certain attributes of the domain objects regarding the information encoded in the policy action.
- Prioritize policies according to their priority attribute.

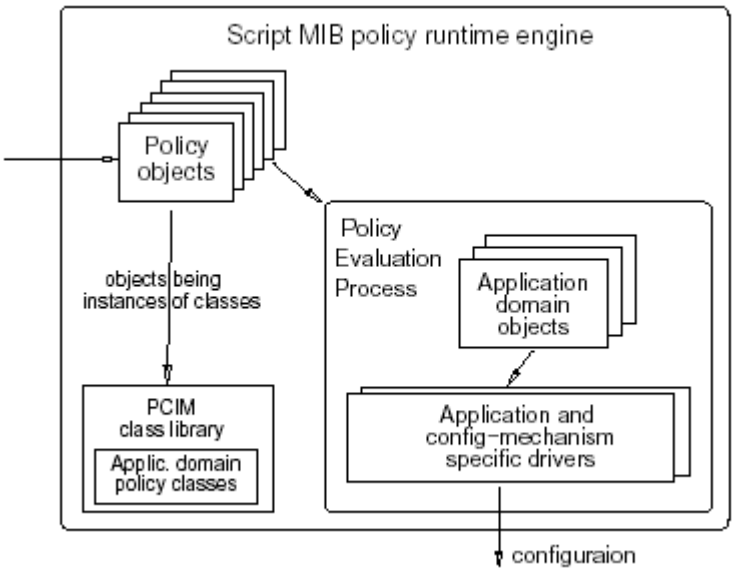

Figure 4.12 Implementation of the policy runtime engine.

4.4 HP OpenView PolicyXpert

PolicyXpert is a policy management system developed by Hewlett Packard [59]. HP OpenView PolicyXpert provides a way to manage the network by authoring and distributing policies, and getting feedback on that policy in the actual network environment.

4.4.1 PolicyXpert System Architecture

There are three major components in the PolicyXpert architecture, Policy Console, Policy Server and Policy Agents. These components are described below:

• The policy console is the user interface where you create and manage policies. The policy definitions specify, in a generalized form, a set of networking services along with the criteria used to regulate the provisioning of and access to those services.

• The policy server stores policies authored in the console as well as status and other information provided by agents. It maintains the full system view of all computing systems, devices, resources, and policies.

Every policy server contains a policy decision point (PDP) component that evaluates policies, their rules, and their targets in order to render policy decisions to any agent that is outsourcing its decision making to the server.

Each primary policy server includes a policy database, or repository. The proprietary-format database is a structured binary file, or set of files, used to store policy, deployment, and status information.

• The policy agents are used to deploy policies to their target resources. The agent informs the primary server about the status of policy deployments and the configuration states of the devices supported by that agent.

As shown in **Figure 4.13** configuration agent and outsourcing PEP run in the managed device. Communication between the policy console, the policy server, and the agent uses the Common Open Policy Service (COPS) standard.

It is supported to define a single policy and deploy it to heterogeneous devices throughout the network, across multiple kinds of elements from multiple vendors. Any managed object supporting one or more of the Quality of Service methods (Class of Service, Bandwidth, RSVP) can be integrated with PolicyXpert, then managed using defined policies.

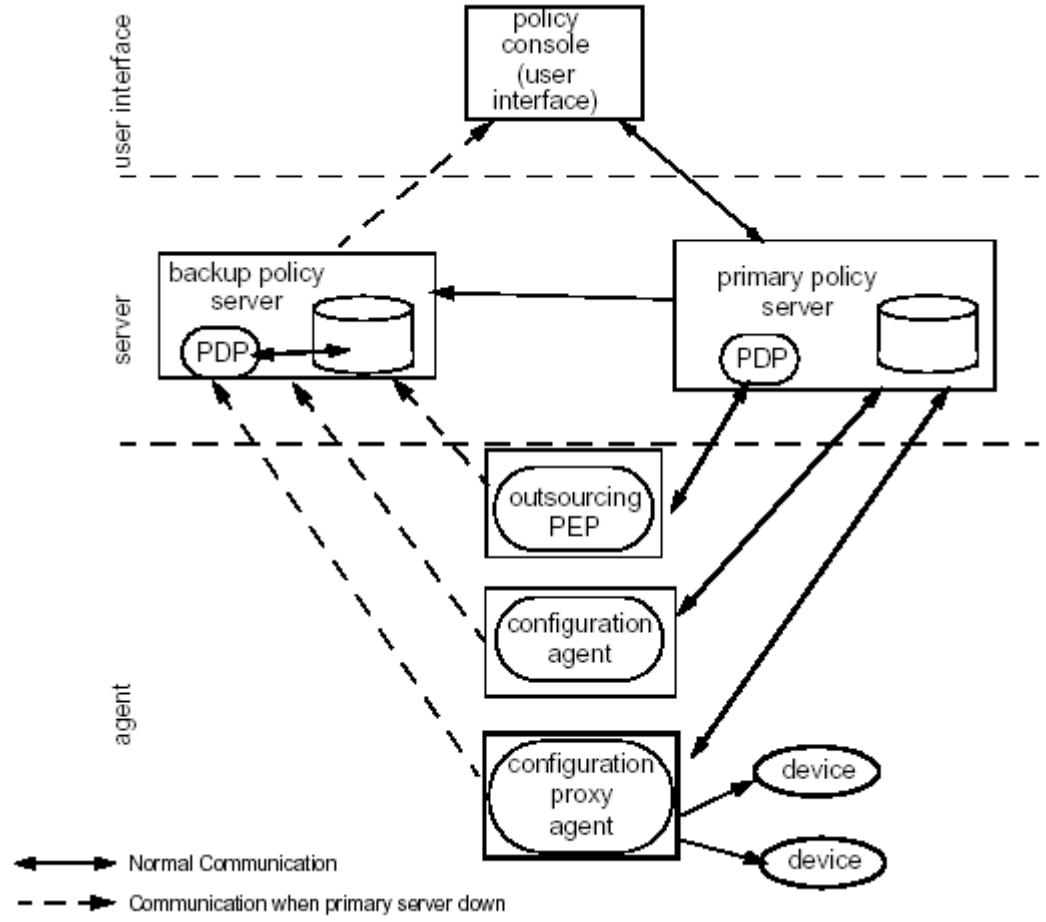

Figure 4.13 PolicyXpert architecture.

4.4.2 Policy Enforcement Points

Policy agents are the integration point between PolicyXpert and managed elements (hardware or software). A policy agent converts policy information to device configuration and commands. The agents enforce the specific behaviors that a resource must provide according to the policy definitions. Agents also provide information to the policy server and console about policy-manageable elements [60].

There are two kinds of policy agents in PolicyXpert [60]:

- Policy Enforcement Point: A Policy Enforcement Point (PEP) is an agent that operates within a managed device and outsources the policy decision making process to the policy server. It only implements, or enforces, the policy decisions made by the server. A PEP translates the policy decision into the device specific actions to carry out the decision. Currently, COPS-RSVP enforcement points are the only type of PEP supported by PolicyXpert [62].

- Policy Configuration Agent: A policy configuration agent receives generalized policy definitions from a policy server and translates the policy information into device-specific configuration and/or commands. In essence, a policy configuration agent performs both the decision making and enforcement processes associated with a policy. The configuration agent provides information to the policy server about what policy-manageable elements exist on the managed device or system, and what types of policies the agent and/or device can support.

A policy configuration agent may reside directly on the managed device or system, or it may reside on some other system. If the configuration agent does not reside directly on the managed device, it is often referred to as a proxy configuration agent. Whether the policy configuration agent is embedded in the managed device or is a proxy agent, it communicates with the PolicyXpert server using the General Policy Interface (GPI) API/library [60].

The configuration agent uses the policy information obtained via the GPI to generate and/or update the appropriate device configurations for the resources that it supports. Agent-specific modules translate the policy information into the appropriate device-specific configurations. In addition, these agent-specific modules perform the actual provisioning of the devices being managed. The configuration operations are done using whatever protocols and/or mechanisms are available for the specific type of device. Common protocols to configure devices via proxy include telnet/CLI, tftp and SNMP.

4.5 IPHIGHWAY

The IPHighway Open Policy System (OPS) [61] shown in **Figure 4.14** uses centralized management architecture with distributed execution. OPS has one Policy Administrator that manages the distributed policy servers (PDPs) and multiple Policy Servers.

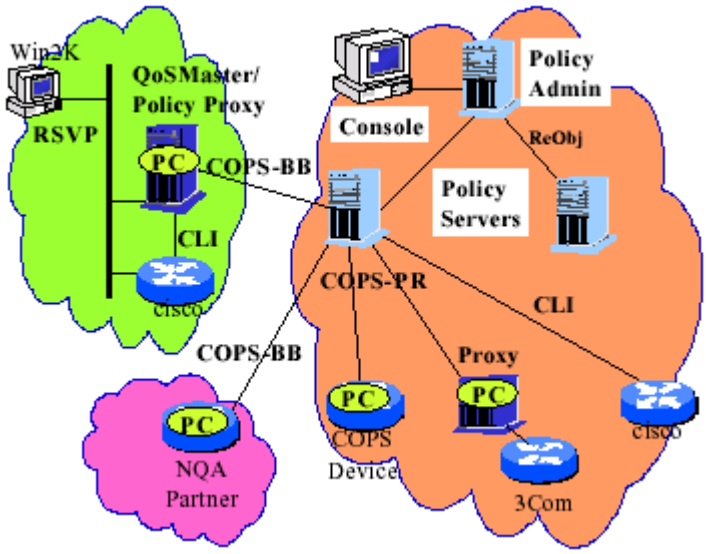

Figure 4.14 Open Policy System (OPS)

4.5.1 OPS System Architecture

The components that make up the Open Policy System are:

- Policy Administrator: Central manager of the distributed policy servers
- Policy Console: User/Administrator GUI for the Policy Administrator
- Policy Server: Real-time, distributed servers controlling policy decision
- Policy Gateway (QoSMaster): LAN/WAN (Wide Area Network) policy coordinator. QoSMaster provides congestion management and policy-based network control for timesensitive applications, important users, and other services typically of interest to enterprise network operators.
- Policy Client/Proxy: Software for adding policy control capabilities to PEPs

Policy Administrator can distribute policies to multiple Policy Servers, and correlate system feedback from those servers for monitoring and accounting purposes.

Multiple policy servers may receive a policy rule if it is to be applied at several locations in the network. The support for multiple Policy Servers, the automatic recognition of new servers, and the automatic distribution of policy throughout the network are features of OPS that afford it the capability to scale to virtually any size network.

The Policy Servers can automatically discover network devices that may be controlled by the policy system, and reports them to the Policy Administrator when initializing. Once discovered, the devices are assigned to one of the OPS Policy Servers in the system. Network devices may also be manually entered into the system, and assigned to a Policy Server.

4.5.1.1 OPS Policy Enforcement Point

The Policy Server directly interacts with network devices, and acts as the Policy Decision Point (PDP), providing policy control for multiple network devices such as routers, switches, VPN devices, firewalls, and other Policy Enforcement Points (PEP). When a Policy Server is activated in the OPS-controlled network, the Policy Administrator to which it is assigned automatically recognizes it, and distributes the appropriate policy rules to it.

The Policy Server converts policy rules received from the Policy Administrator into a form that can be understood by the devices it controls. That form may be device specific, such as CLI commands for a Cisco router; it may also be a standards-based protocol message, such as COPS. The Policy Server gathers status information from the devices it controls, and uses this feedback when making policy decisions related to the allocation of network resources and the assignment of those resources. The feedback is passed up to the Policy Administrator, ensuring that a complete and consistent view of the network is maintained.

IPHighway's Policy Client controls virtually any device that is capable of enforcing QoS policy rules, allowing the device to participate in policy-based networking. When integrated into a network device, the Policy Client provides the device with an interface to the OPS Policy Server, allowing the device to be managed by the policy based management system. The current version of Policy Client is based on the Common Open Policy Service (COPS) IETF standards track protocols, developed in the RAP working group.

5 POLICE POLICY ENFORCEMENT POINT

Police Policy Enforcement Point is the system that is responsible for the enforcement of the policies compiled by POLICE Policy Management System. POLICE provides a management console to system administrator to write his policies and compiles these policies. It has a Domain Service to store the policies and domain information. POLICE Policy Server deploys the policies to PPEPs and PPEP performs a set of operations to apply these policies to the managed objects that it controls. The detailed description of POLICE system is given in Chapter 3.

In this section POLICE Policy Enforcement Point architecture, inside modules and operation are described. Also a brief comparison section is included. Comparison is done between the policy enforcement point architectures of Police, Active PBM, Ponder, Script MIB, HP OpenView PolicyXpert and IP Highway OPS. These systems were studied in Chapter 4.

5.1 General Architecture of PPEP

Police Policy Enforcement Point runs within a network device and takes the responsibility of applying the policies sent by the Policy Server and sending information about managed device to Policy Server.

The enforcement of both obligation policies and authorization policies are done by Policy Controller in Police Policy Enforcement system. PEP receives the policy objects from Policy Deployment Service via PDS Agent and Messaging Service (JMS) and Policy Controller enforces these policies after an interaction with Event Service and Resource Controller. Policy Controller applies the obligation policies directly when the constraints are proved but does not apply the authorization policies unless a request comes from the managed nodes via Resource Controller. Policy Controller sends the policies to managed objects via Managed Node Interface.

Components of PPEP of which the general architecture is shown in **Figure 5.1** are as follows:

- Policy Controller
- Policy Deployment Service Agent
- Event Service
- DataBase
- Resource Controller
- Node Management Interface
- Messaging Service
- PPEP Manager

PEP components and concepts used in PEP are defined in the next sections.

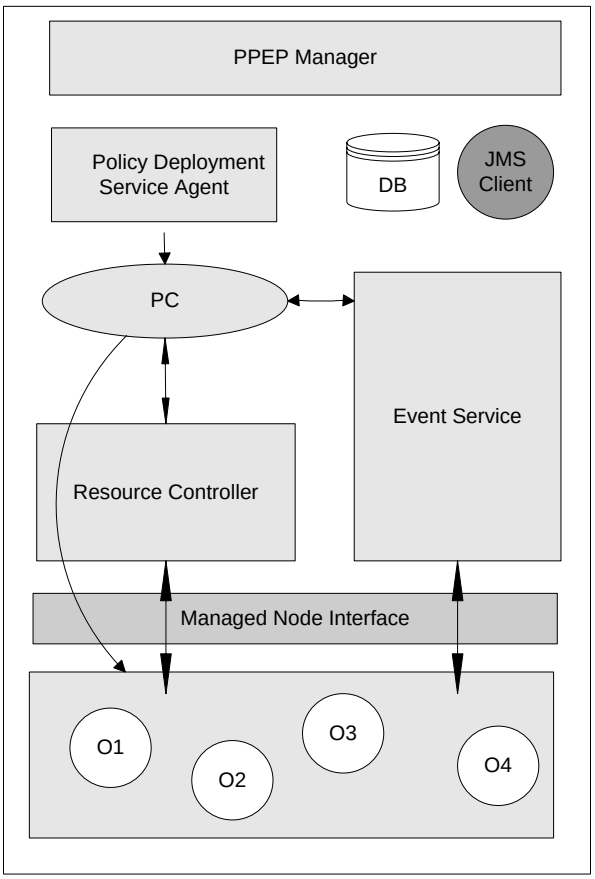

Figure 5.1 POLICE Management System Policy Enforcement Architecture

5.2 PPEP Manager

PPEP manager takes place at the startup of the PPEP. It scans the DataBase tables to initialize the policies and registers to Domain Server.

PPEP firstly directs the CIMOM to get the information about the managed objects and system events from the managed device and sends this information to Domain Server.

PPEP Manager reads all policies at the policy table and sends them to Policy Controller in order to notify the Event Service and makes it possible to apply the policies again, just like receiving them from the Policy Deployment Service.

5.3 Policy Controller

Policy Controller (PC) is the unit which is responsible for applying the policies to the related managed objects in the enforcement point. PC receives the policies from the Policy Deployment Service and stores them in the DataBase in policy table. PC sends the "when constraints" to Event Service and waits for Policy Gauge's result to apply the policy.

PC stores both obligation and authorization policies in the same table and applies every obligation policy to the managed objects when the constraint is proved, for authorization policies PC only changes the state value of the constraint in the policy table when received the result from PG and replies the authorization requests coming from managed nodes via Resource Controller.

5.3.1 Policy Controller Components

5.3.1.1 Policy DataBase

Policy Controller Policy DB is a table in which the Policies received from PDS are stored. Policy DB table fields are:

- "policy name" the primary key field of the table identifies policies and includes the domain path,
- "type" shows the type of the policy; it can take two values,
    - obligation policy
    - authorization policy
- "actor" the object that will process the "action",
- "target" the object that the action will be processed on,
- "action" the action to be taken when the condition policy is proved,

- “condition state” the flag that shows if the condition is proved or not.

The structure of this table is shown in **Table 5.1**. Default value for the state field is (X) meaning that the constraint is not yet proved. After the Event Service returns the prove value this state turns to be (√).

Table 5.1 Policy DB

Policy Name	Type	Actor	Target	Action	Condition State
Domain\Policy	OBL	O1	O5	Method1	X
					X
		. . .	. . .	. . .	. . .

Policy DB is searched by PolicyName value. PolicyName is unique for every policy and it includes the domain path that the policy implemented. The ConditionState field value is changed by Event Service – Policy Gauge process results.

5.3.1.2 Process Policy Module

Process policy module receives the policies from PDS Agent with a parameter telling the action to be taken with that policy. It performs three kinds of actions:

ADD_POLICY: Inserts the policy to PolicyTable and checks the policy constraint, if the constraint is an ABSOLUTE constraint there is no need to send the constraint to Event Service else the constraint structure is sent to Event Service with PolicyName parameter as an identifier. Absolute constraints can have two different types: Never or Always. If the policy has a never constraint that means this action is prohibited else if it is always constraint that means this policy will be applied without any checks.

DELETE_POLICY: Removes the policy entry from the PolicyTable and from Event Service DB tables and the policy gauge module and the consumer threads are stopped.

UPDATE_POLICY: Removes the policy entry from the PolicyTable and from Event Service DB tables and add the policy just as a new one.

5.3.1.3 Apply Policy Module

Apply policy module is triggered by Policy Gauge modules. Event Service creates a policy gauge for each policy entry and policy gauge checks the constraint state value of the related policy in a loop while the policy is in the Policy DB. Policy gauge triggers the policy controller in two states:

- DOWN: The constraint value changes from '1' to '0'
- UP: The constraint value changes from '0' to '1'

Apply policy module performs different actions according to the UP/DOWN parameter. For example if the policy is as follows

```
inst mustdo filterIP {
  actor    a = /Router/router1/routingDeamon;
  target   /Router/router1/eth0;
  action   down(port);
  when     throughput>50Mbit;
```

If the throughput > 50 Mbit policy gauge informs the PC with UP parameter and PC forces the managed object to close eth0 port. When the throughput value decreases policy gauge knows this and informs the PC again but this time with DOWN parameter and PC forces the managed object to open the port.

If the constraint value of the policy is an ABSOLUTE constraint than the ES doesn't process this constraint and doesn't create policy gauge for this policy, in this situation process policy module itself calls the apply policy module and the value of this policy's constraint state doesn't change until this policy updated.

5.3.2 Operation of Policy Controller

PDS agent on PPEP receives the policies from the Policy Deployment Service of the Management System via JMS and decides if the received message includes a new policy to insert, or an old policy to delete or update. PDS sends the policy to PC with an action parameter. According to the action parameter which shows what to do with this policy, PC either writes the policy to Policy DB or deletes it from the Policy DB or updates the policy entry in the policy DB.

For all policies which have a condition having a type except for ABSOLUTE_CONSTRAINT, PC sends a request to Event Service in order to be notified when the condition is set. Event Service writes these constraints in ConditionTable and registers to CIMOM Event Trigger for the events in the

constraint, and creates a Policy Gauge for each policy. Policy Gauge creates event consumer thread and notifies the policy controller in every action of the state value. When PC receives the notification it writes (√) or (X) to the "condition state" and applies the policy.

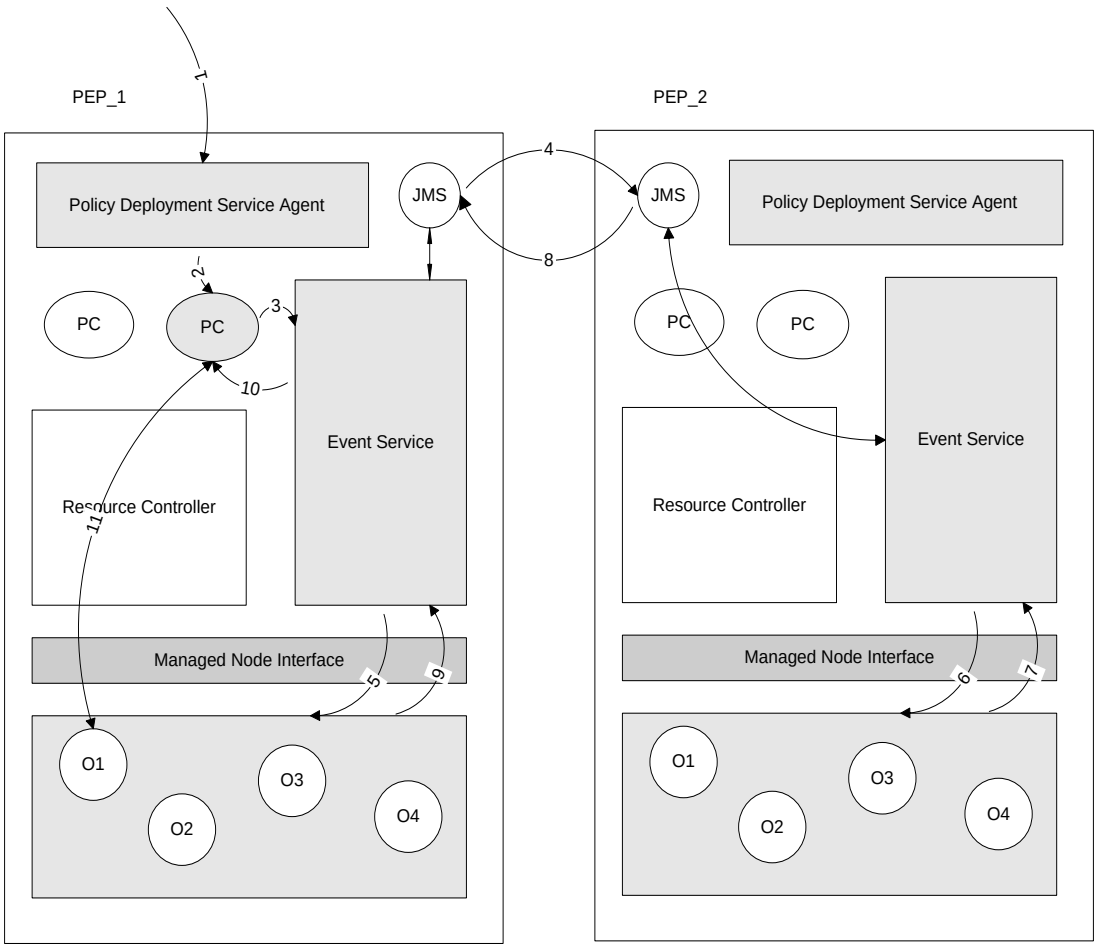

Figure 5.2 Enforcement of an obligation policy

Figure 5.2 shows the policy enforcement of an obligation policy specified as:

```
mustdo policyOBL {
                    actor  : O1 ;
                    target : O3;
                    action : O3.Action1;
                    when   : O7.Event1 && O4.Event2;
}
```

PEP_1 receives the policy from Policy Server (1) via PDS agent and sends it to PC (2), PC processes the policy and sends the constraint object to ES if needed (3), ES sends Inter Node Event Messages to remote PPEPs if there is a remote related event (4). Both the local (5) and remote (6) PPEPs wait for the event to occur in their

systems and when the remote one receives the event (7) it sends a reply message to local PPEP (8). The local PPEP also receives and processes the local events (9) and triggers the PC (10), PC applies the obligation policy (11).

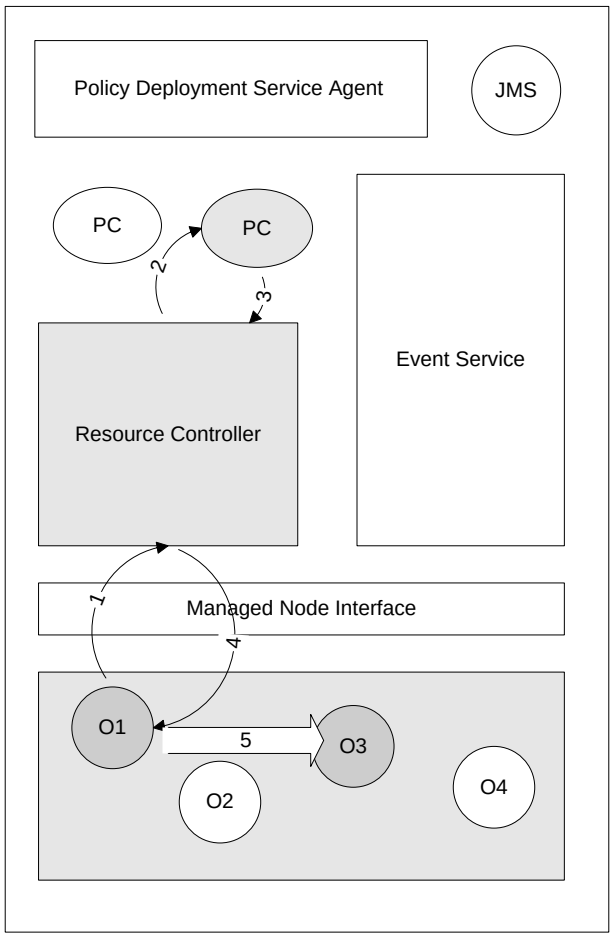

Figure 5.3 Enforcement of an authorization policy

Figure 5.3 shows the policy enforcement of an authorization policy specified as

cando policyAUTH {
actor : O1 ;
target : O3;
action : O3.Action1;
when : O7.Event1 & O4.Event2;
}

PC stores the authorization policies in the policy table and updates the state values when the constraint value changes. When RC receives an authorization request from a managed object (1), it redirects it to the related PC (2) and PC searches it at the policy table. PC sends the reply (3) according to the policy search to RC and RC informs the managed node (4).

For a better performance, policy objects can have another field to specify policy type:

- Continuous: when the "condition state" turns to (√) a method is called; when it turns to (X) another method is called; and this continuous until the policy is unloaded or deleted.
- Once: PC waits for the Event Service for "condition state" to turn to (√) and calls the related method then PC deletes the Policy Record from Event Service DB and from its own DB.

But these two types of policies are not yet implemented.

5.4 Event Service

The Event Service collects and composes events from the managed nodes and from remote Event Services of other PPEPs, and forwards them to the registered PCs which are executing policies. The ES is an integrated service spread through the distributed system nodes via the Inter-Node Event Bus. Event Service has following modules:

- Condition Table: The constraints received from PCs and their state values are stored in condition table.
- Policy Gauge: Watches the state value of the constraints on condition table, and triggers PC.
- Event Service Manager: Receives the constraints from PCs and processes them.
- Inter-Node Event Collector: Used for communication with remote ESs.

5.4.1 Operation of Event Service

The block diagram showing the interaction of an ES with other Police PEP modules is shown in **Figure 5.4**. The Event Service is in interaction with Policy Controller, JMS client and Node Management Interface.

Policy Controller registers one constraint to Event Service for each obligation and authorization policy of which the constraint type is not an ABSOLUTE Constraint. When a constraint is received, ES parses it and divides it into separate events, inserts

a record to the Condition Table and creates a Policy Gauge for each. Policy Gauge evaluates the constraint state, updates the Condition Table and triggers PC for every change in the value of the state of the constraint. The events can belong to local managed objects or to a remote PPEP's managed objects.

Policy Gauge creates Event Consumers (ECs) for the constraint and each Event Consumer watches an event or a pseudo event or a value of composite events. The pseudo-events in the constraint section of a policy are created to observe attributes of the managed system's objects and occur when these attributes reach at specified value ranges.

For the local ones event ECs register to Event Trigger in NMI and NMI triggers EC when the event occurs. For remote ones, EC registers to Inter Node Event Collector and Inter Node Event Collector sends a message to the remote PPEP via Inter node Event Bus and triggers the EC when an event occurred message is received. For local pseudo events, EC polls the CIMOM periodically. If the pseudo event belongs to a remote node EC registers to Inter Node Event Collector and Inter Node Event Collector holds a table, and writes every value of the needed property to this table as it receives messages from remote. ECs poll this table for pseudo events just like polling CIMOM.

When a PPEP receives an Event Message from a remote PPEP, it creates a Remote Poller for pseudo events and this module polls the CIMOM periodically and sends a reply message to remote for each value. For events it directly registers to Event Trigger in NMI and sends a reply message when it is triggered by NMI for the event occurrence.

Unlike other policy specification languages, Police uses event-triggers for both authorization and obligation policies. This makes it possible to change access rights according to the system events as well. Another superiority that this design provides is homogenous enforcement for all types of policies.

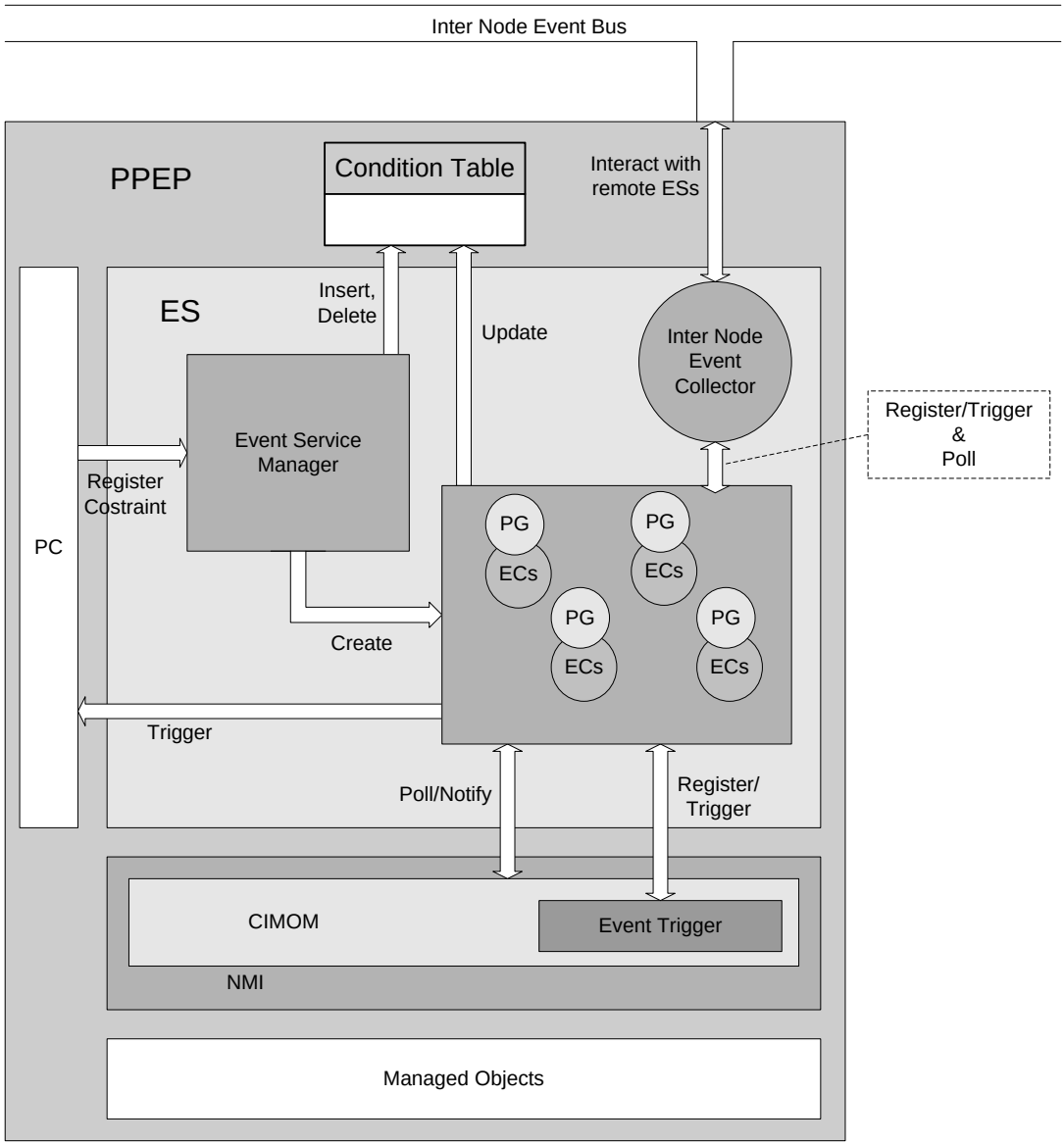

Figure 5.4 Block Diagram of ES in Police PEP

5.4.2 Condition Table

Condition table is used to store the constraint value of each policy and its current state value. The structure of the condition table is shown in **Table 5.2**. Fields of the table are:

- "policy name" the name of the related policy, unique for each policy
- "condition" the object that keeps the constraint value of the policy
- "state" the value that shows if the constraint is proved or not

The primary key of the table is "policy name" field.

Table 5.2 Condition Table

Policy Name	Condition	State
PolicyA	Event1&&Event4	X
PolicyB	Event2||Event3	X
. . .	. . .	. . .

The event processor unit inserts one entry to the condition table on each time it receives a policy from policy controller and it deletes the related entry when received a delete message from PC. Policy gauge updates the condition table when it finds out that the constraint is proved. PG decides this after communicating with NMI or remote PPEPs.

5.4.3 Policy Gauge

Policy Gauge is the module which watches the policy constraint's state value and informs the Policy Controller when the state value changes.

After a PC receives a policy from the PDS, it parses the policy and decides if the policy constraint needs to be registered to Event Service or not. There are 8 types of constraints, these are:

- COMPOSITE_CONSTRAINT
- TIME_CONSTRAINT
- SUBJECT_CONSTRAINT
- TARGET_CONSTRAINT
- ACTION_CONSTRAINT
- EVENT_CONSTRAINT
- SYSTEM_CONSTRAINT
- ABSOLUTE_CONSTRAINT

There is no need to register to ES for an ABSOLUTE_CONSTRAINT type constraint, because this type of constraint has two options NEVER, ALWAYS. For the other types of constraints PC registers to Event Service and Event Service Manager creates one Policy Gauge for each policy constraint it receives from PC. This Policy Gauge stays alive while the policy entry is in Policy Controller Policy

Table. This PG evaluates the constraint state continuously while the policy is not deleted.

Each policy gauge module works on one policy constraint. This constraint can contain multiple events and multiple logical relations. The constraint object is parsed into its events and pseudo events in Policy Gauge and each event forms an Event Consumer. Event Consumers construct a hierarchic structure described in 5.4.3.2 under Policy Gauge and the Event Consumer at the highest level notifies the PG of changed in the state value.

The constraint sections of the Police policies are implemented as event-statements composed of system-events and pseudo-events. In order to express event statements, complicated event expressions can be written by the help of Policy Definition Language (PDL) [16], defining several operators to derive complex events from primitive events.

The pseudo-events are created to observe attributes of the managed system's objects and occur when a specified value range of these attributes are reached. Also, the sophisticated time period conditions can be implemented as pseudo-events. For example, the statements like "object1.attribute2 > 200", "Date > 10.12.2003" are considered as pseudo-events. The pseudo-events compose complex event statements with the system events in the constraint section of the policies. The constraint part of our policy specification syntax can include such statements as below:

when (**$event_1$ && $event_2$**) || (Object5.attribute4 =12) || (Date > 10.12.2003)

when **$event_1$** && (Object5.attribute3 > 200) AND ^**$event_2$**

when (Object15.attribute1 >= Constant.SUSPEND) || **$event_1^3$**

When the "state" value changes, the PG triggers the related PC. So, there is no need the PC to evaluate its policies whether the conditions are met to execute them, as the ES performs this task for PCs.

A policy gauge output example is given in **Figure 5.5**. PG triggers the PC at t0, t1, t2, t3. PC performs different actions for the DOWN and UP sides of the gauge output. According to this figure, within the time intervals of [t0-t1] and [t2-t3], policy gauge is active. PC stores the gauge outputs in its own policy table.

While a gauge is active, if related policy is an authorization policy, the actors can perform actions if they attempt, otherwise nothing will happen. In case of the obligation policy, the activation of the gauge causes the PC to enforce the actors of the policy to do actions of the policy and deactivation of gauges stop the actors. When triggering the PC, Policy Gauge also sends information about UP or DOWN side.

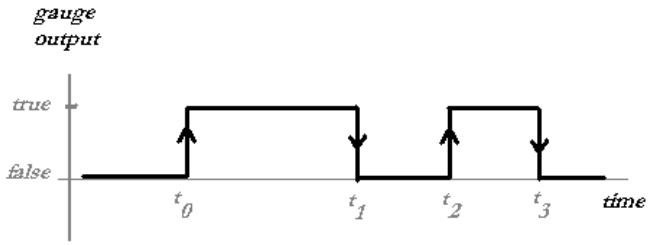

Figure 5.5 A sample Policy Gauge output

5.4.3.1 Policy Gauge Evaluation Method

A Policy Gauge creates a tree of event consumer objects which corresponds to the runtime event specification created by the compiler. Each of the event consumers is responsible for a node of the event tree. Event consumers appearing as leaf nodes in the tree correspond to single events and are responsible for registering to Event Trigger of CIMOM or Inter Node Event Collector (INEC) and are responsible for polling CIMOM or INEC as described in 5.4.3.2, to receive notification for those events.

Composite event consumers are responsible for enforcing the semantics of the event operator they are representing.

The composite consumer for the → operator will only send a notification to the next level if it receives a notification for event at the left side of the operator before the event at the right side.

The composite consumer for the **&&** (AND) operator will notify the upper side consumer after it receives a notification from both the left and right side events of the operator.

Similarly, the composite consumer for the || (OR) operator will notify the upper side consumer after it receives a notification from either the left or right side events of the operator.

5.4.3.2 Event Consumers

Event Consumers Tree is a part of the Policy Gauge which evaluates the whole constraint and notifies the Policy Gauge when the result changes. Event Consumers are created recursively in a hierarchic structure from top to bottom. They evaluate the constraint recursively from bottom to top. This structure is shown in **Figure 5.6** for the constraint:

((Event_1 AND PseudoEvent_1) OR ((Event_2 AND PseudoEvent_2) OR Event_3))

Firstly the constraint is divided into two parts at the first level and the operator between them is placed at the top "OR":

1- ((Event_1 AND PseudoEvent_1)

2- ((((Event_2 AND PseudoEvent_2) OR Event_3)

Then these two parts are divided into their events and their operators are placed in their positions and second level is constructed:

1.1- Event_1

1.2- PseudoEvent_1

2.1- (Event_2 AND PseudoEvent_2)

2.2- Event_3

2.1 is still a composite constraint so it should be divided once more, this is the third and the last level:

2.1.1 Event_2

2.1.2 PseudoEvent_2

The consumers which do not have a sub-consumer starts working to learn their event's or pseudo event's values from Local CIMOM or Remote CIMOM.

The Pseudo Event Consumers firstly decides if the pseudo event is local or remote. If it is local consumer polls the CIMOM periodically, else consumer registers to Inter Node Event Collector and polls it.

The Event Consumers firstly decides if the event is local or remote. If it is local consumer registers to Event Trigger in CIMOM and waits for being triggered, else consumer registers to Inter Node Event Collector and waits trigger from there.

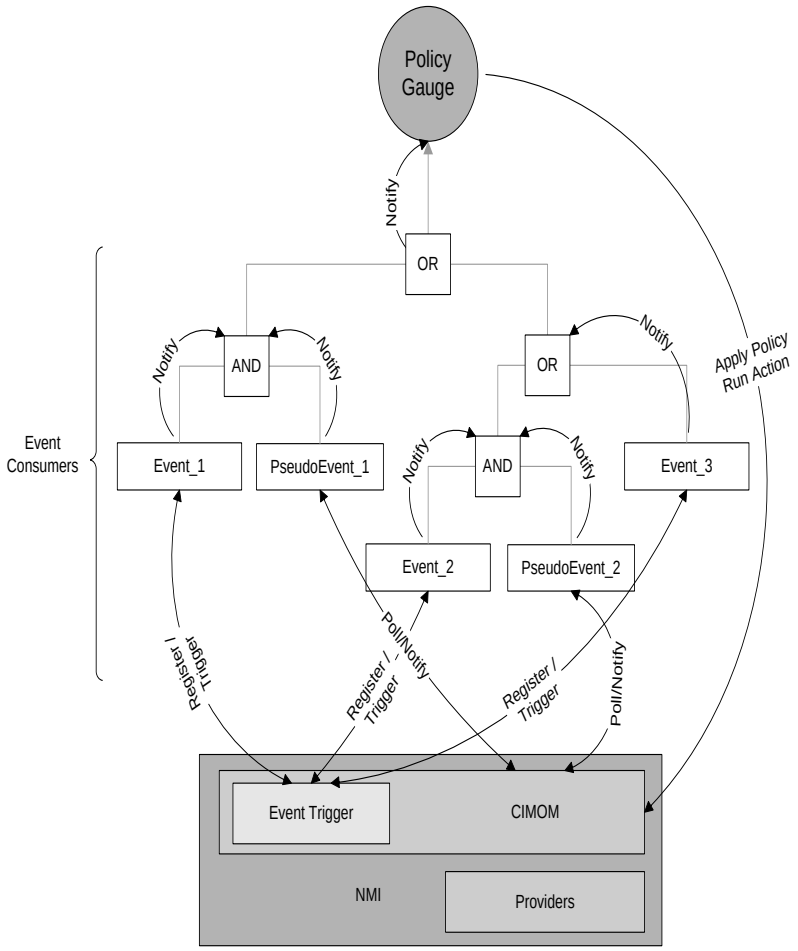

Figure 5.6 Event Consumers schema for constraint:
((Event_1 AND PseudoEvent_1) OR ((Event_2 AND PseudoEvent_2) OR Event_3))

5.4.4 Event Service Manager

Event Service Manager is the main of ES that organizes every other modules operation. It interacts with Police Controller and Condition Table and creates the Policy Gauges as shown in Figure 5.4.

Policy Controller registers one constraint to Event Service Manager for each obligation and authorization policy of which the constraint type is not an ABSOLUTE Constraint. When a constraint is received, ESP inserts a record to the Condition Table and creates a Policy Gauge for it. Then the responsibility of triggering PC is Policy Gauge's told in 5.4.3.

5.4.5 Inter-Node Event Collector

Inter-Node Event Collector interacts with the other Event Services over the Inter-Node Event Bus. Inter-Node Event Bus carries the messages by JMS using a JMS Server. POLICE management system has a JMS Server for events and every PPEP has a JMS client. These clients send messages to collect data from the other ESs and also send the events they collected from their own nodes in order to inform other ESs.

The message formats are shown in **Figure 5.7** and **Figure 5.8**.

Message Type	PPEP_ID Source	PPEP_ID Responder	Managed Object property/Event	Event Type

Figure 5.7 Request Message

Message Type	PPEP_ID Responder	PPEP_ID Source	Managed Object property/Event	Event Type	Event State / Value	TimeStamp

Figure 5.8 Reply Message

Inter Node Event collector operates as shown in **Figure 5.9**. Event Consumers can handle 4 different types of events:

- Local events
- Local pseudo events
- Remote events
- Remote pseudo events

For remote events and remote pseudo events, ES has to communicate with the remote PEPP's ES. Event Consumer registers to INEC for these events and waits for being triggered for remote events, but polls INEC for remote pseudo events, because INEC holds the received values of remote in a table in it. The aim of this approach is not to change the general working principal of ECs. Just as polling local pseudo events from CIMOM, it can poll remote ones from INEC.

If INEC receives a request message from another PPEP's INEC, it just registers to CIMOM – Event Trigger module for "events" and when CIMOM triggers it, it sends a reply message to remote in order to trigger remote. If the received message

includes a pseudo event INEC creates a Remote Poller thread to poll CIMOM for this property and sends periodic messages to remote to notify about this value.

Inter Node Event Collector operation steps shown in **Figure 5.9** are explained below. These steps start at the point where PC sends the constraints to ES and ES creates Policy Gauges and Event Consumers. The constraint of PG1 includes two pseudo events (1 local, 1 remote) and one local event, PG2 includes two events (1 local, 1 remote).

- Step1: PG1 consumers start polling CIMOM for local pseudo event and receive the values periodically.
- Step2: PG1 consumers register to local CIMOM Event Trigger for local event.
- Step3: PG1 is triggered by CIMOM Event Trigger when the related event occurs.
- Step4: PG1 consumers register to INEC for remote pseudo event and start polling INEC.
- Step5: INEC sends a request message (of which the format is shown in **Figure 5.7**) to remote INEC for the need in step 5 and creates a register for this value to let the Event Consumer polling.
- Step6: Remote INEC receives the message sent in step 5 and creates a Remote Poller thread to poll the CIMOM.
- Step7: Remote Poller starts polling CIMOM.
- Step8: Remote Poller notifies remote INEC about the value it polls periodically.
- Step9: Remote INEC sends reply messages periodically to inform PPEP 1 INEC.
- Step10: PG2 Event Consumers register to INEC for remote event.
- Step11: INEC sends a request message to remote.
- Step12: Remote receives the message sent at step 11 and registers to its CIMOM Event Trigger.

- Step13: Remote CIMOM triggers the remote INEC when the event occurs.
- Step14: Remote INEC sends a reply message when it is triggered by CIMOM Event Trigger.
- Step15: PPEP 1 INEC triggers the related Event Consumer when it receives the reply message sent in step 14.

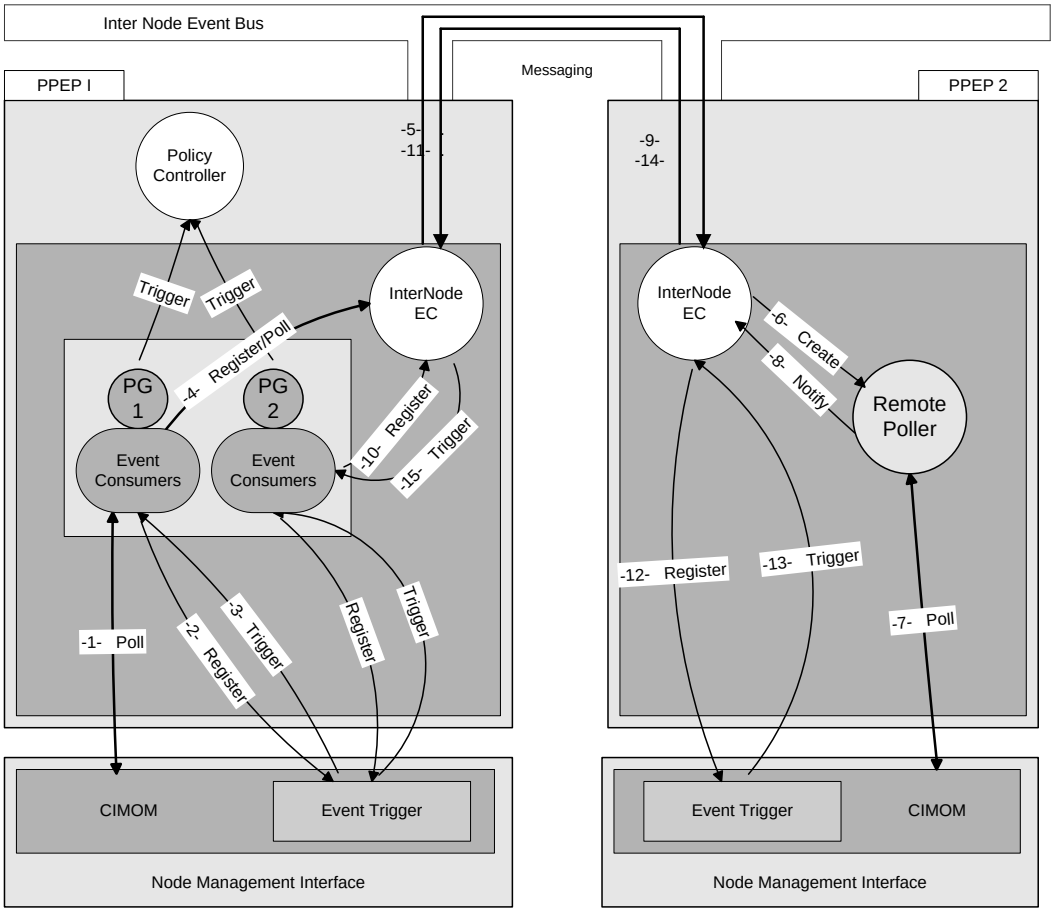

Figure 5.9 Inter Node Event Collector messaging via Inter Node Event Bus

5.4.5.1 Remote Poller

Remote Poller is the module which polls a single pseudo event's value requested by a remote PPEP. It notifies the INEC every time it receives a different value from CIMOM. It polls the CIMOM periodically.

Event Service creates one Remote Poller thread for each request message coming from remote for pseudo events.

Creating an event consumer at remote PPEP can be an alternative solution. At this solution, ES sends the whole event consumer to the remote PPEP for remote attributes and the PPEP receiving the message creates a thread for each consumer request and evaluates its value. Remote PPEP sends the value of this consumer to the requesting PPEP when its value changes.

The advantage of the first solution shows itself when multiple ECs watch the value of the same attribute. Otherwise the second solution seems to have better performance.

5.5 Resource Controller

Resource Controller is the unit which acts as an interface between the PC and the managed objects for the authorization policies.

Normally, each actor is in a scope of one PC, which enforces all policies defined for that actor object. A PC controls all the actors at its location and enforces all policies relating to its actors. Being primarily a stationary agent, each PC normally enforces many policies and controls the behaviors of many different actors in the domain for which this PC is responsible.

In the PEP architecture, the Resource Controller intercepts all access requests of the actors and redirects them to concerned PC to check whether the access is permitted. The block diagram showing the interactions of RC with other system units is in **Figure 5.10**. When an object attempts to execute an operation on another object, first it should have the permission to perform this action. Authorization policies are stored in Policy Controller Policy Table.

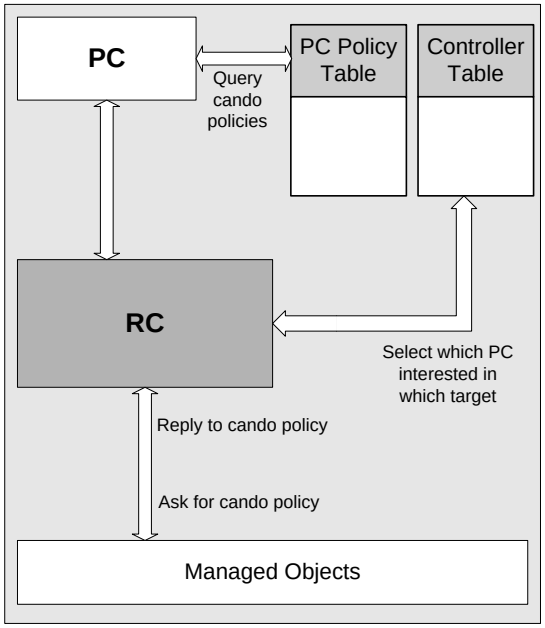

Figure 5.10 Block diagram of Resource Controller

5.5.1 Controller Table

Controller table is used to store the relations between the Policy Controllers and target objects. Each entry in the table shows the PC that controls the managed object named "object". The structure of the controller table is shown in **Table 5.3**. Fields of the table are:

- "object" the name of the target object
- "policy controller" the id of the PC that controls the target.

The primary key of the table is "target" field.

Table 5.3 Controller Table

Object	Policy controller
TargetOBJ1	PC_1
TargetOBJ2	PC_2
. . .	. . .

5.5.2 Operation of Resource Controller

When a managed object wants to perform an action on another managed object, first it has to get permission for this from the Policy controller. The managed object communicates with the Resource Controller over the Node Management Interface

and queries the RC via a message containing the target object and the action to be performed. This operation is shown in **Figure 5.3**.

RC searches the target on the Controller Table and finds out the related PC. Then the RC sends a request to that PC containing Actor (the object that sent the request to RC), Target and Action. When PC receives this request it searches the related policy in the Policy Table, and returns the "State" value to RC. RC informs the actor node about the result. If the requested action is authorized the object can execute the action else it cannot.

5.6 Policy Deployment Service Agent

Policy Deployment Service Agent receives the policy objects sent by Policy Deployment Service via JMS Server, and sends them to the related Policy Controller. PDS agent on PPEP has an interface to JMS Server on the Management Center. It receives the policies sent by the Server via Messaging Service and processes these messages according to the Message Type. There are three types of messages:

- ADD_POLICY
- UPDATE_POLICY
- DELETE_POLICY

PDS Agents search the Resource Controller Table to find out the PC to which the policy will be sent. RC Table is searched by the object field (primary key of the table) with actor name in the policy. PDS agent then sends the policy structure to Policy Controller with a parameter to specify what to do with the policy; add, delete or update. After receiving a message PDS Agent sends an acknowledgment message to the Server. **Figure 5.11** shows the flow diagram of PDS Agent on PPEP.

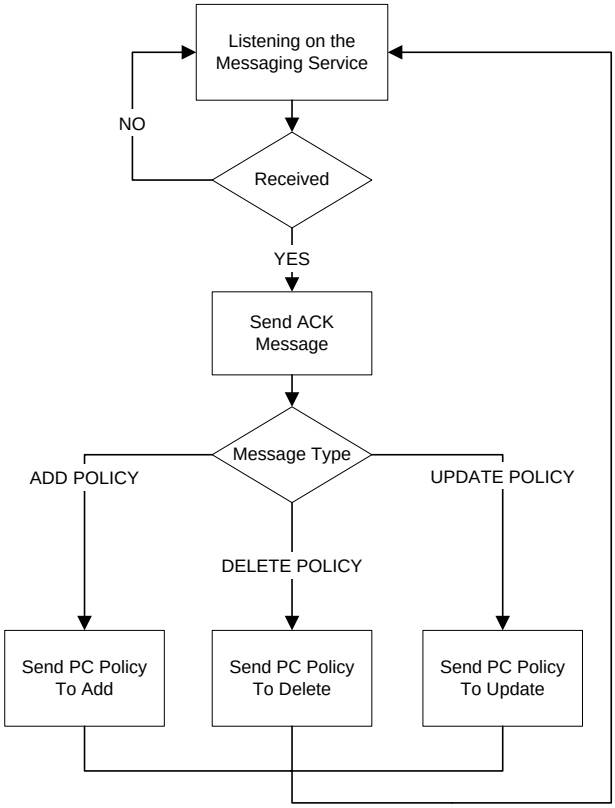

Figure 5.11 Flow diagram of PDS Agent on PPEP

5.7 Node Management Interface

The Police Node Management Interface provides a standard interface between the Police PEP and the embedded hardware and software. In this way, the management activities are executed on representatives of the real managed objects and thus a uniform access on object interfaces during execution of policies is possible. When accessing the modules' interfaces through the mediation layer to perform operations specified in the action list, the method calls are transformed into a series of system specific function calls on the actual managed object.

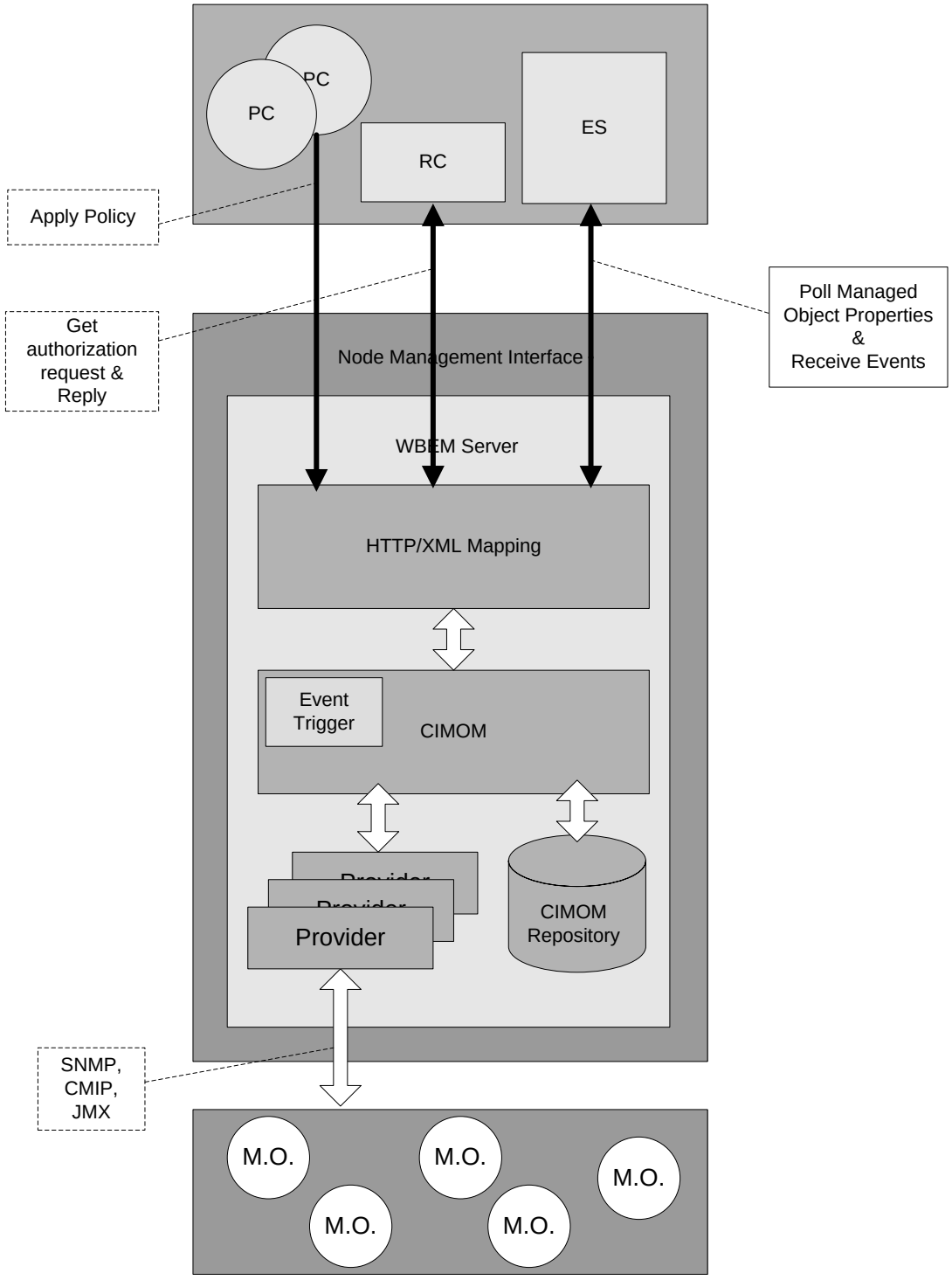

Figure 5.12 CIM Instrumentation of a Managed Node.

In order to implement a flexible node management interface that allows the Police PEPs to enforce policies in an independent way of the underlying implementation of the managed system, Common Information Model Object Manager (CIMOM) based approach defined in the Web-Based Enterprise Management (WBEM) technologies of the DMTF [63] is used in architecture.

5.7.1 Operation of NMI

In each NMI, a CIMOM is employed to serve the requests of Police PEP for the data about the managed objects. The NMI includes a standard and a product-specific element schema loaded into the CIMOM repository. The NMI includes also native provider interface to communicate with the managed node via native providers such as SNMP, CMIP, JMX providers.

WBEM uses a client server model as shown in **Figure 5.12**. Policy Controllers, Resource Controller and Event Service act as a WBEM Client and send the messages to CIMOM via HTTP/XML Mapping Module. This module converts the messages coming from PCs, ES or RC to a format that the CIMOM can understand and converts the messages of CIMOM in a format that is suitable to send via HTTP.

PC sends policy apply requests to CIMOM, RC gets the authorization requests from the managed objects and replies them, and ES asks some values of managed objects and receives the events that occurs.

CIMOM receives the requests coming from these clients and firstly looks at the CIMOM Repository for static information such as schema definitions or static instances, if it cannot find the information it needs then forwards the request to the provider. CIMOM knows which provider is responsible for which managed object from the element schemas.

Consumers waiting for an event registers to Event Trigger in CIMOM. Event Trigger holds a list of events for which the consumers registered and the consumers that are registered for each event. Event Trigger receives the events from providers and notifies the consumers registered to it.

Providers communicate with the managed nodes, collect any data from these nodes. In Police PEP, providers are also responsible for enforcing the actions in the policies to the managed objects and detecting and forwarding events to the management application via CIMOM.

If the data received from the managed object is in a native format, provider maps it to CIM classes before passing to CIMOM. In the same way if the managed node that CIMOM wants to learn requires data in native format, provider maps the CIM classes received from CIMOM to native format and then passes to the managed object.

CIMOM Repository stores the CIM class definitions and dynamic data. The definitions are converted from a text MOF file into CIM Repository formatted data.

5.8 DataBase

Police Policy Enforcement Point needs storage for the information of some relations between the inside components, the received policies, system events and the constraint objects of the policies. This storage is designed as a DB containing a table for each data class.

There are 3 different tables to store:

- Policy Controller – Actor relation
- Policies to be applied
- Constraint (condition) Objects that the Policy Controller waits for to apply a policy

The implementation details of the DB and the tables are given in 6.2.

5.9 Messaging Service

In Police PBM framework PPEP communicates with Police Policy Deployment Service and also with other PPEPs via Inter Node Event Bus. It receives polices from PDS and sends state and registration information to PDS. PPEPs communicate with each other to request or reply event states. For these operations we chose a messaging system of which the implementation details are given in 6.1.

5.10 Domain Server Registration

Police Policy Server receives the policies from management console, compiles and stores them in Domain Service Policy Repository and sends them to Police PEPs. Domain Service also includes Domain Objects Repository. Domain Service stores the information about which Policy PEP is responsible of which managed objects. This information is needed when Police Server is about to send a policy to PPEP.

After compiling a policy, Police Server queries the Domain Service to with the Policy/Actor parameter and learns the PPEP that is responsible for this Actor.

Domain Service can get this information in two ways:

- Manually : Police Server administrator can put this information to server manually via server console.
- Automatically : PPEP registers itself to server on startup.

On startup, PPEP sends the information of the managed objects that it is responsible for to Police Server via Inter Node Deployment and Communication Bus. This information includes the names of the Managed Objects and the events that can occur in the managed system.

5.11 Comparison with Other Management Systems' PEPs

In this section some properties of Police PEP is compared with the same properties of the PEPs of the frameworks studied in Chapter 4. The list of all compared properties is given in A.1. Some fields of this table are explained more detailed in the following paragraphs.

Police Policy Enforcement Points communicate with each other via Inter Node Event Bus. This communication aims to collect data and event from remote nodes. PPEP can learn a property of a remote managed node or can be triggered when a specific event occurs at a remote node. Except for Police, this kind of a communication is supported by only Active PBM. Active PBM PDPs communicate each other via PDP Inter-Domain capsules when a PDP has to take a decision related to an inter-domain connection. PEPs can only communicate with PDPs, Active PBM uses either PEP capsules or PDP capsules for this communication.

Unlike other systems Police and Ponder store the policies at a local policy table at the PEP. Other systems store their policies at PDP. Using PDP for all device policies may hold the PDPs busy, in a distributed system it is preferable to distribute policies too, although some say it is better to keep PEP simpler. Another advantage of local storage shows itself in "cando" (authorization) policies, when managed device wants to get authorization it doesn't need to ask to PDP which a remote stand alone unit but it just asks to PEP agent.

Police PEP supports event triggering for both obligation and authorization policies while Ponder only supports obligation for policies. This makes it possible to change access rights according to the system events as well. Active PBM also works over event triggering method and stores the events in PDP, this may cause network to be

busier. Other systems do not use event triggering. Active PBM sends every event to PDP and executes for every kind of policy. In Ponder events trigger the Event Publisher at the management center. In Police events trigger the Event Services in the PEPs.

Police PEPs register to Policy Server on startup and IpHighway OPS PEP is discovered by Policy Server, but other systems do not support such a property.

None of the PBM systems that we studied include a security module. When we look at the academic products, only in Script MIB security module is applicable. HP OpenView and IpHighway do not implemented security module, too. SNMP security based on SNMPv3 and the user based security model and on the view based access control model is fully applicable to the Script MIB. Police uses JMS for communication between distributed units and JMS, itself has a security module so it is also applicable to Police.

The superiority of our enforcement architecture is catching prohibited actions at the source, before these actions consume some system resources. In Ponder, for example, actions are checked at target-side to determine whether the actors are allowed to perform them. In contrast, the Police philosophy is to assign a policeman for each actor in the system, not to set up defense lines around the resources against any actor. We control the actions at their sources before they are started.

Both obligation and authorization policies are distributed in the same uniform way to the PCs near the actors that it makes our architecture simpler.

Police architecture is suitable to create multiple PCs for different types of policies or targets etc. This can help e.g. to separate obligation and authorization policy controllers and run these two paralleled.

6 IMPLEMENTATION OF PPEP

In order to show the feasibility of an effective policy enforcement system based on the idea presented in this thesis, a prototype implementation has been developed to work inter operable with Police PBM framework.

The node management interface is simulated but the other modules are mostly implemented. Another part of the simulation is managed node. Police PEP cannot work with a real network device yet.

For simplicity only one Policy Controller is created in this prototype, in the next versions multi PC should be implemented and tested. It needs measuring in order to see the advantages or disadvantages.

The application is developed on Windows 2000 operating system with Borland JBuilder 9 compiler. Open JMS messaging service and MySQL database is used to provide a working environment to Police PEP. In this chapter, details of implementation and evaluation of PPEP according to this implementation is given.

6.1 JMS

Rather than communicating directly with each other, the components in an application based around a message service send messages to a message server. The message server, in turn, delivers the messages to the specified recipients [64].

The common building block of a messaging service is the message. Messages are events, requests, and replies that are created by and delivered to enterprise applications. Messages contain formatted data with specific business meanings.

Open JMS is used as messaging service in Police PBM system and Police PEP. The Java Message Service (JMS) API [64] products are the most scalable and mostly used messaging middleware solutions. Messaging systems provide a way to exchange data asynchronously and the JMS API allows a java application to connect to these systems. Once connected, an application can use the facilities of the

underlying enterprise messaging system to create messages and communicate asynchronously with one or more peer applications. Specific goals of the JMS API specification are to:

- Provide a single, unified message API
- Provide an API suitable for the creation of messages that match the format used by existing, non-JMS applications
- Support the development of heterogeneous applications that span operating systems, platforms, architectures, and computer languages
- Support messages that contain serialized Java objects
- Support messages that contain XML pages

6.2 DataBase

We used MySQL open source database for database implementation of POLICE Policy Enforcement Point application and MyODBC 3.51 as ODBC Driver. Also MySQL Control Center is used to create our PPEP tables and observe debug tests.

MySQL, a popular Open Source SQL database, is developed and provided by MySQL AB. MySQL AB is a commercial company that builds its business providing services around the MySQL database.

MySQL Connector/ODBC (also known as MyODBC) allows connecting to a MySQL database server using the ODBC database API on all Microsoft Windows and most Unix platforms, including through such applications and programming environments such as Microsoft Access, Microsoft Excel, and Borland Delphi.

6.2.1 Table Structures

Tables used to store the Police PEP data structures are created via MySQL Control Center. Each PPEP connects to a DB having the same name with itself and uses the username password written in the configuration file to establish this connection. Also the tables could be created automatically at initialization of the PPEP once at the startup; we chose to create them manually for this demo version. The structures of the tables are given in **Table 6.1** and **Table 6.2** with the primary key fields for each one.

Table 6.1 Policy Controller Policy Table

policycontroller_policytable		
Policy Name	Primary Key	Varchar
Type		Int
Actor		Varchar
Target		Varchar
Action		Varchar
Condition State		Int

Table 6.2 Condition Table

conditiontable		
Policy Name	Primary Key	Varchar
Condition		Blob
State		Int

6.3 Simulation of Managed Device and NMI

PPEP runs just like an agent with each device to be managed by Police. Node Management Interface is the interface between managed device and PPEP. NMI provides flexibility to PPEP to work with any kind of network device. By the help of an appropriate NMI implementation PPEP can control every device.

NMI applies the policies sent by Policy Controller, receives the events from managed system, receives any authorization request from managed nodes and replies to them after getting a result from Policy controller. It interacts with Policy Controller directly for applying obligation policies and via Resource Controller for authorization policies.

In PPEP implementation, neither a specific network device used nor NMI implemented but both units are simulated in one Java class “Simulator”. Simulator provides the following to help us watching the general operation of PPEP:

- Create events: Simulator gets the event names from the user as an input and sends it directly to Node Event Collector.
- Event Trigger: When an event is created Simulator triggers the related PPEP modules (consumers). It provides an interface to PPEP to register when an

event is waited. Consumers register to Event Trigger for an event and when the event is created, Event Trigger triggers the registered consumers.

- Apply obligation polices: When Policy Controller makes a decision to apply an obligation policy it triggers the Simulator about applying a policy and Simulator warns the user of the policy to be applied showing an information message.
- Creates authorization request: Simulator can create an authorization request. User should give the actor node, target node and the action as input. Simulator sends this request to Resource Manager and waits for the reply. When received the reply it shows it to the user on an information message.

6.4 Evaluation of PPEP

To measure the time needed to load policies and process the system events a simple test environment is set. Server ran on a stand alone PC and one another PC runs two PPEPs.

Tests a re performed deploying 100 policies and then creating 100 events on PPEPs. It is observed that, PPEP can process 45 policies in1 minute and 27 events in 1 minute.

Measurements depend on lots of parameter; these are CPU, RAM, DB Access, debugging information, logs, etc. Of course one important parameter is network performance.

7 CONCLUSION

In this thesis we presented general approach to Policy Enforcement of Policy Based Networks, a Policy Enforcement Point "Police PEP" and made a research of some other PBM systems' PEPs.

Police Policy Based Management system is studied and an enforcement point architecture is designed to work with Police, Police PEP.

Active PMB, Ponder, Script MIB, HPOpenView and IPHihgway OPS are researched and the general architectures and policy enforcement mechanisms of these systems are studied and classified. Our PEP mechanism is compared with these systems.

With Police all the managed objects in a distributed system are organized into hierarchical domains. Policies are then written in terms of domains. This means that there is normally no need to write policies specific to an object. Instead, whenever an object is added to a domain, the policies that apply to that domain will automatically be applied to the object, until either the object is removed from the domain or the policy is disabled.

Main advantages of Police PPEP are the communication between PEPs, the enforcement of authorization policies and the event triggering method.

Both the authorization and the obligation policies are enforced in the same way. Unlike any other system in Police the authorization policies can also be triggered by any system event.

Police PEPs can learn the properties of other managed devices properties or the events that occur in other systems via Inter Node Event Messaging and can use this information for either obligation policies or authorization policies. For this messaging PPEPs do not need to communicate with Policy Server (PDP), this protects PDPs from being overloaded.

Most of the other systems keep their policies only in policy repository and send them to PEP when the policy is about to be applied while this time interval the PDPs check

the constraints of policies and decide when the policy will be applied. After PDP decides to apply the policy it sends it to PEP. In Police the policies are sent to PPEP directly when they are created and PPEP makes the decision to apply the policy and PPEP checks the constraint of the policy. So PPEP does not need to send the events occur in its managed device to PDP.

The Node Management Interface of PPEP is not designed in detail yet and not implemented, it is just simulated. As a future work it will be designed and implemented by Hasan Er at Sakarya University.

REFERENCES

[1] http://whatis.techtarget.com, 2003.

[2] **Haixin, D. and Jianping, W.**, 2000. Policy-based Access Control Framework for Large Networks, IEEE International Conference on Networks (ICON'00).

[3] **Westerinen A., Schnizlein J., Strassner J., Scherling M., Quinn B., Herzog S., Huynh A., Carlson M., Perry J. and Waldbusser S.**, 2001. Terminology for Policy-Based Management, RFC 3198.

[4] **Moore B., Ellesson E., Strassner J. and Westerinen A.**, 2001. Policy Core Information Model -- Version 1 Specification, RFC 3060.

[5] **Dinesh, C. Verma**, 2000. Policy-Based Networking, New Riders, Indianapolis.

[6] **Yavatkar R., Pendarakis D. and Guerin R.**, 2000. A Framework for Policy-based Admission Control, RFC 2753.

[7] **O'Sullivan, T. C.**, 1971. Telnet Protocol, RFC 0137.

[8] **Whitehead, E., Irvine, UC and Murata, M.**, 1998. XML Media Types, RFC 2376.

[9] **Ryan, V., Lee, R. and Seligman, S.**, 1999. Schema for Representing CORBA Object References in an LDAP Directory, RFC 2714.

[10] **Wahl, M., Howes, T. and Kille, S.**, 1997. Lightweight Directory Access Protocol (v3), RFC 2251.

[11] **Strassner, J., Ellesson, E. and Moore, B.**, 1999. Eds., Policy Framework Core Information Model, Internet draft, draft-ietf-policy-core-schema-02.txt, Feb. 1999.

[12] **CIM**, 1999. Common Information Model (CIM) Specification version 2.2, Distributed Management Task Force, Inc. (DMTF), available on WWW from www.dmtf.org/spec/cims.html, 14 June 1999.

[13] **Radisi I.**, 2002. Using Policy-Based Concepts to Provide Service Oriented Accounting Management, In Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS2002),2002

[14] **Sun WBEM**, SDK -http://wbemservices.sourceforge.net/

[15] **T. Hamada, P. Czezowski, T. Chujo**, 2000. Policy-based Management for Enterprise and Carrier IP Networking, FUJITSU Sci. Tech. J.,36,2, 128-139

[16] **A.Moridero, K. Murano**,"The Network Paradigm of the 21th Century and its Key Technologies", IEEE Computer Networks Nov 2000.

[17] **Fonseca, M., Agoulmine, N., and Cherkaoui, O.**, 2001. Active Networks as a Flexible Approach to deploy QoS Policy Based Management, HP

Openview University Association 8th Annual Workshop, HP-OVUA http://www.hpovua.org/, Berlin, Germany, June 24 – 27, 2001

[18] **V. A. Pham and A. Karmouch**, Mobile Software Agents: An overview, IEEE Communication Magazine, pp 26-37, 1998

[19] **Damianou, N., Dulay, N., Lupu, E., and Sloman, M.**, 2000. Ponder: A Language for Specifying Security and Management Policies for Distributed Systems. The Language Specification - Version 2.3. Research Report DoC 2000/1, Imperial College of Science Technology and Medicine, Department of Computing, London, 20 October 2000.

[20] **Rosen, E., Viswanathan, A., Callon, R.**, 1999. Multiprotocol Label Switching Architecture", IETF <draft-ietf-mpls-arch-06.txt>, August 1999

[21] **Lupu, E. C.**, 1998. A Role-Based Framework for Distributed Systems Management. PhD Thesis, Department of Computing, Imperial College. London, U. K., July 1998

[22] **DENWEB**, http://www.denwebdns.com, 2003

[23] **Damianou, N. C., T. Tonouchi, Dulay, N., Lupu, E., Sloman, M.**, 2002. Tools for Domain-based Policy Management of Distributed Systems. In Proceedings of the Network Operations and Management Symposium (NOMS 2002), Frorence, Italy, 15-19 April 2002

[24] **P. Martinez**, et al., 2002. Using the Script MIB for Policy-based Configuration Management, In Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS2002), 2002

[25] **Muruganantha, N. and Lutfiyya, H.**, 2003. Policy Specification and Architecture for Quality of Service Management, Integrated Management Symposium (IM03), 2003

[26] **Yixin Diao, Frank Eskesen**, et al., 2003. Generic On-Line Discovery of Quantitative Models for Service Level Management, Integrated Management Symposium (IM03), 2003

[27] **DMTF** official web site, www.dmtf.org

[28] **SNIA CIMOM**, http://www.snia.org

[29] **Verma, D. C.** (2001). Policy-Based Networking: Architecture and Algorithms, New Riders Publishing, 2000

[30] **Weinberg, N.**, 1999. Policy-based networks: Easier said than done, Network World Magazine, 08/23/99

[31] **Caruso, J.**, 1999. Policy-based management ain't what it used to be, Network World Magazine, available on WWW from http://www.nwfusion.com/archive/1999b/0412edit.html 04/12/99

[32] **HP OpenView** "PolicyXpert Primer on Policy-based Network Management", Edition 1 Windows NT® operating system Manufacturing Part Number: J1360-90001 October 1999

[33] **Dursun, T., Nasır, D., and Örencik, B.**, 2004. Relation Based Policy Conflict Detection, Proceedings of the 3rd Asia Pacific Int'l. Symp. on

Information Technology (3rd APIS), İTÜ, İstanbul, Jan. 13-14, 2004, pp.192-198.

[34] **Dursun, T., and Örencik, B.**, 2003. Dağıtık Politika Çakışması Çözme Yaklaşımı, 1.Türkiye Ulusal Yazılım Sempozyumu (UYMS03), Ege Üniversitesi, İzmir, Ekim, 2003

[35] **Dursun, T., and Örencik, B.**, 2003. Politika Tabanlı Sistemlerde Yeni Bir Dağıtık Çakışma Sezme Yaklaşımı, 20.Türkiye Ulusal Bilişim Kurultayı (TBK2003), Eylül 2-5, 2003, Taksim, İstanbul

[36] **Dursun, T., and Örencik, B.**, 2003. POLICE: A Novel Policy Framework, Lecture Notes in Computer Science, LNCS 2869, Nov. 2003, pp. 819-827

[37] **Gagnon, E.**, 1998. SableCC, An Object-Oriented Compiler Framework. MSc Thesis, School of Computer Science, McGill University. Montreal, Canada, March 1998

[38] **Moffett, J. D. and Sloman, M. S.**, 1993. Policy Hierarchies for Distributed Systems Management. IEEE JSAC Special Issue on Network Management, 11(9), 1404-1414

[39] **Lupu, E. C. and Sloman, M. S.**, 1999. Conflicts in Policy-Based Distributed Systems Management, In IEEE Transactions on Software Engineering - Special Issue on Inconsistency Management, 25(6), pp. 852-869

[40] **Samarati, P. and Vimercati, S.**, 2000. Access Control: Policies, Models, and Mechanisms. In Foundations of Security Analysis and Design (Tutorial Lectures). R. Focardi and R. Gorrieri eds, Springer -Verlag, pp. 137-196, September 2000

[41] **OMG**, 1999, Object Management Group, Object Constraint Language Specification, version 1.3, Chapter 7 in OMG Unified Modelling Language Version 1.3, June 1999

[42] **Damianou, N. C., Dulay, N., Lupu, E., Sloman, M.**, 2000. Managing Security in Object-based Distributed Systems using Ponder. In Proceedings of the 6th Open European Summer School (Eunice 2000), Enchede, The Netherlands, 13-15 September 2000

[43] **Strassner, J. and Schleimer, S.**, 1998. Policy Framework Definition Language (version 00), IETF Internet draft work in progress, available from http://www.ietf.org, 17 November 1997

[44] **Lobo, J., Bhatia, R. and Naqvi, S.**,1999. A Policy Description Language. In Proceedings of the Sixteenth National Conference on Artificial Intelligence Eleventh Innovative Applications of AI Conference, Orlando, Florida, USA, 18-22 July 1999

[45] **Gamma, E. and Helm, R.**, 1995. Design Patterns, Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995

[46] **Sloman, M. and Twidle, K.**, 1994. Domains: A Framework for Structuring Management Policy. In Chapter 16 in Network and Distributed Systems Management (Sloman, 1994ed), pp. 433-453

[47] **Yialelis, N.**, 1996. Domain-Based Security for Distributed Object Systems. PhD Thesis, Department of Computing, Imperial College. London, U. K., August 1996

[48] **Dulay, N., Lupu, E., Sloman, M., Damianou, N.**, May 2001. A Policy Deployment Model for the Ponder Language. An extended version of paper in Proc. IEEE/IFIP International Symposium on Integrated Network Management (IM'2001), Seattle.

[49] **Achir, N., Fonseca, M. S. P., Doudane, Y. M. Ghamri, Agoulmine, N. and Mehaoua, A.**, October 2000. Active Networking System Evaluation: A Practical Experience, MoMuc.

[50] **Agoulmine, N., Dragan, D., Gringel, T., Halt, J., Rosa, E. and Tschichholz M.**, 2000. Trouble Management for Multimedia Services in Multi-Provider Environments, International Journal on Network and Service Management, Wiley publisher.

[51] **TeleManagement Forum**, June 2001. SLA Management Handbook, GB 917 v1.5, Public Evaluation Version

[52] **Marshal, A.**, May 2000. Dynamic Network Adaptation Techniques for Improving Service Quality, Networking, Paris.

[53] **Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z. and Weiss,W.**, December 1998. An Architecture for Differentiated Services, RFC 2475, Network Working Group, IETF

[54] **Wetherall, D.**, April 1998. ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols, IEEE OPENARCH'98, San Francisco.

[55] **Biswas, Jit**. Application Programming Interfaces for Networks (A White Paper prepared by the Working Group for IEEE 1520 on Application Programming Interface Networks), http://www.iss.nus.sg/IEEEPIN.

[56] **Damianou, Nicodemos C.**, February 2002. A Policy Framework for Management of Distributed Systems, Imperial College of Science, Technology and Medicine University of Department of Computing, London.

[57] **Strauß F.**, May 2001. Concept of a Script MIB based Policy Management System, Computer Science Department, Technical University, Braunschweig, Germany

[59] **Snir, Y., Ramberg, Y., Strassner, J. and Cohen, R.**, November 2001. Policy QoS Information Model, Internet Draft <draft-ietf-policy-qos-info-model-04.txt>, Cisco Systems, Ntear LLC.

[59] **Hewlett Packard**, October 1999. OpenView PolicyXpert User's Guide Edition 1, Manufacturing Part Number: J1360-90002.

[60] **Hewlett Packard**, October 1999. OpenView PolicyXpert Developer's Guide Edition 1, Manufacturing Part Number: J1360-90003.

[61] **IPHighWay**, January 2000. Policy Based Networking Products, Design and Architecture Volume#3.

[62] **IETF**, http://www.ietf.org, 2003

[63] **Sun WBEM SDK,** http://wbemservices.sourceforge.net/

[64] **JavaWorld**, http://www.javaworld.com, 2003

APPENDIX A.1 Comparison Table

AUTOBIOGRAPHY

Delal Nasır was born in Muğla-Milas on 8-September-1978. She has graduated from Soke Hilmi Firat Anatolia High School at 1996 and started studying at Istanbul Technical University at the same year. She has completed bachelor of Control and Computer Engineering at 2000 and she has been working in TUBITAK-UEKAE for 3.5 years.