Policy-based Quality of Service Management in IP Networks

Paris Flegkas

Submitted for the Degree of
Doctor of Philosophy
from the
University of Surrey

Centre for Communication Systems Research
School of Electronics and Physical Sciences
University of Surrey
Guildford, Surrey GU2 7XH, UK

April 2005

Summary

For years the Internet networking community has been struggling to develop ways to manage networks. Initial attempts brought mechanisms and protocols that focused on managing and configuring individual networking devices i.e. the Simple Network Management Protocol (SNMP). This model worked well in early deployments of IP management systems for local and metropolitan area networks but now, with the evolution of Quality of Service (QoS) models such as the Differentiated Services (DiffServ) framework, the complexity and overhead of operating and administrating networks increases enormously. There is also a need to be able to program management systems and network components to adapt to emerging requirements and subsequently be able to dynamically change the behaviour of the whole system to support modified or additional functionality. The emerging Policy-based Network Management paradigm claims to be a solution to these requirements. Policy-based Management can guide the behaviour of a network or distributed system through high-level declarative directives that are dynamically introduced, checked for consistency, refined and evaluated, resulting typically in a series of low-level actions.

The objective of this thesis is to investigate the application of Policy-based Management in the context of QoS Management of IP DiffServ Networks. By using policies as a means for building extensible hierarchical management systems, we propose a novel Policy-based QoS management architecture and specify the relevant policies that can drive its behaviour dynamically, providing a holistic approach to the area of policies for QoS Management.

We first present our view on policies as a means of extending the functionality of management systems dynamically, in conjunction with pre-existing "hard-wired" management logic and provide a generic framework for their application to hierarchical distributed management systems. The programmability aspect of policies is an issue which has not been properly addressed in the literature and constitutes an important contribution of this thesis. We then propose a single architecture for managing an IP DiffServ network, identifying the required functional components and their interactions addressing both service management and resource provisioning (traffic engineering) aspects of QoS Management. The design of the architecture caters for both offline and dynamic operation. Furthermore, we identify the parameters of the functional components of the architecture that are influenced by policies and present an object oriented representation of those policies based on the Policy Core Information Model (PCIM). This work differentiates from relevant work on QoS policies since it addresses the area of QoS Management in its totality

defining policies related to service management and traffic engineering both at a network and element management level.

Finally, we validate the proposed policy-based framework by presenting a detailed description of the design and implementation of the components of the policy management sub-system needed to be deployed in order to make our system policy-driven and present examples of QoS policies describing their transformation from their definition by the operator until their enforcement.

Key words: Policy-based Management, QoS Management, IP Differentiated Services, Service Management, Traffic Engineering

Email: p.flegkas@surrey.ac.uk

WWW: http://www.ee.surrey.ac.uk/Personal/P.Flegkas

Dedicated to

Acknowledgments

Contents

List of Figures

List of Tables

Abbreviations

AAA	Authentication, Authorization and Accounting
AF	Assured Forwarding
API	Application Programming Interface
AS	Autonomous System
ATM	Asynchronous Transfer Mode
BA	Behaviour Aggregate
BB	Bandwidth Broker
BE	Best Effort
BML	Business Management Layer
bps	Bits per second
CBR	Constant Bit Rate
CIM	Common Information Model
CLI	Command Line Interface
CNF	Conjunctive Normal Form
CO	Computational Object
COD	Code on Demand
COPS	Common Open Policy Service
COPS-PR	Common Open Policy Service for Provisioning
CORBA	Common Object Request Broker Architecture
CoS	Class of Service
DE	Default
DiffServ	Differentiated Services
DIT	Directory Information Tree
DMTF	Distributed Management Task Force
DN	Distinguished Name

DNF	Disjunctive Normal Form
DNS	Domain Name System
DOT	Distributed Object Technology
DRsM	Dynamic Resource Management
DRtM	Dynamic Route Management
DSCP	DiffServ Codepoint
DTD	Document Type Definition
EF	Expedited Forwarding
EML	Element Management Layer
EWMA	Exponentially Weighted Moving Average
FB	Functional Block
FEC	Forward Equivalent Class
FlowDes	Flow Description
GUI	Graphical User Interface
IDL	Interface Definition Language
IETF	Internet Engineering Task Force
IN	Intelligent Network
IntServ	Integrated Services
IP	Internet Protocol
ISP	Internet Service Provider
LDAP	Lightweight Directory Access Protocol
LPDP	Local Policy Decision Point
LSP	Label Switched Path
LSR	Label Switched Router
M'O	Managing Object
MA	Mobile Agent
MbD	Management by Delegation
MDT	Mean Down Time

MF	Multi-Field
MO	Managed Object
MPLS	Multi-Protocol Label Switching
MR	Monitoring Repository
MTU	Maximum Transfer Unit
ND	Network Dimensioning
NMC	Network Management Centre
NML	Network Management Layer
NR	Network Repository
NS	Network Simulator
NSM	Network and Service Monitor
OA	Ordered Aggregate
OSI-SM	Open Systems Interconnection Systems Management
PC	Policy Consumer
PCIM	Policy Core Information Model
PCIMe	Policy Core Information Model Extensions
PDB	Per-Domain Behaviour
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PHB	Per-Hop Behaviour
PIB	Policy Information Base
PMA	Policy Management Agent
PMT	Policy Management Tool
PPL	Path-based Policy Language
PR	Policy Repository
QC	QoS Class
QoS	Quality of Service
QPIM	QoS Policy Information Model

RAB	Resource Availability Buffer
RAM	Resource Availability Matrix
RAP	Resource Allocation Protocol
RDN	Relative Distinguished Name
RED	Random Early Detection
REV	Remote Evaluation
RPC	Resource Provisioning Cycle
RSVP	Resource Reservation Protocol
SLA	Service Level Agreement
SLS	Service Level Specifications
SLS-I	SLS Invocation
SLS-S	SLS Subscription
SM	Service Manager
SML	Service Management Layer
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
SR	Service Repository
TC	Traffic Conformance
TCS	Traffic Conditioning Specification
TE	Traffic Engineering
TF	Traffic Forecast
TMN	Telecommunications Management Network
TOS	Type of Service
TT	Traffic Trunk
TTR	Time to Repair
VBR	Variable Bit Rate
VLL	Virtual Leased Line
VoIP	Voice over IP

VP	Virtual Path
VPC	Virtual Path Connection
VW	Virtual Wire
WFQ	Weighted Fair Queuing
WG	Working Group

Publications

Journal Papers

1. M. P. Howarth, M. Boucadair, **P. Flegkas**, N. Wang, G. Pavlou, P. Morand, T. Coadic, D. Griffin, A. Asgari, P. Georgatsos, *End-to-end quality of service provisioning through Inter-provider traffic engineering*, to appear in Computer Communications, Special Issue on End-to-End QoS, 2005

2. M. P. Howarth, **P. Flegkas**, G. Pavlou, N. Wang, P. Trimintzios, H. Asgari, D. Griffin, J. Griem, M. Boucadair, P. Morand, P. Georgatsos, *Provisioning for Inter-domain Quality of Service: the MESCAL Approach*, to appear in IEEE Communications, 2005

3. G. Pavlou, **P. Flegkas**, S. Gouveris, A. Liotta, *On Management Technologies and the Potential of Web Services,* IEEE Communications, special issue on XML-based Management of Networks and Services, Vol. 42, No. 7, pp. 58-66, IEEE, July 2004

4. P. Trimintzios, G. Pavlou, **P. Flegkas**, P. Georgatsos, E. Mykoniati, *Service-driven Traffic Engineering for Intra-domain Quality of Service Management*, IEEE Network, special issue on Network Management of Multi-service, Multimedia, IP-based Networks, Vol. 17, No. 3, pp. 29-36, May/June 2003

5. P. Trimintzios, T. Bauge, G. Pavlou, **P.Flegkas**, R. Egan, *Quality of Service Provisioning through Traffic Engineering with Applicability to IP-based Production Networks*, Computer Communications, special issue on Performance Evaluation of IP Networks and Services, Vol. 26, Issue 8, Elsevier Science Publishers, pp. 845-860, May 2003

6. **P. Flegkas**, P. Trimintzios, G. Pavlou, *A Policy-based Quality of Service Management System for IP Differentiated Services Networks*, IEEE Network, special issue on Policy Based Networking, Vol. 16, No. 2, pp. 50-56, IEEE, March/April 2002

7. P. Trimintzios, I. Andrikopoulos, G. Pavlou, **P. Flegkas**, D. Griffin, P. Georgatsos, D. Goderis, Y. T'Joens, L. Georgiadis, C. Jacquenet, R. Egan, *A Management and Control Architecture for Providing IP Differentiated Services in MPLS-based Networks*, in IEEE Communications, special issue in IP Operations and Management, Vol. 39, No. 5, pp. 80-88, May 2001

Peer-reviewed Conference Papers

1. M. Charalambides, **P. Flegkas**, G. Pavlou, A. Bandara, E. Lupu, A. Russo, N. Dulay, M. Sloman, *Policy Conflict Analysis for Quality of Service Management*, to appear in the proceedings of the IEEE Workshop on Policy for Distributed Systems and Networks (Policy 2005), Stockholm, Sweden, IEEE, June 2005

2. J. R. Loyola, J. Serrat, M. Charalambides, **P. Flegkas**, G. Pavlou, A. L. Lafuente, *Using Linear Temporal Model Checking for Goal-Oriented Policy Refinement Frameworks*, to appear in the proceedings of the IEEE Workshop on Policy for Distributed Systems and Networks (Policy 2005), Stockholm, Sweden, IEEE, June 2005

3. A. Bandara, E. Lupu, A. Russo, N. Dulay, M. Sloman, **P. Flegkas**, M. Charalambides, G. Pavlou, *Policy Refinement for DiffServ Quality of Service Management*, to appear in the proceedings of the IEEE/IFIP Integrated Management Symposium (IM'2005), Nice, France, IEEE, May 2005

4. M. Howarth, **P. Flegkas**, G. Pavlou, P. Trimintzios, H. Asgari, D. Griffin, J. Griem, M. Boucadair, P. Georgatsos, *An Architectural Framework for Inter-domain Quality of Service Provisioning*, to appear as a poster in the Proceedings of the IEEE/IFIP Integrated Management Symposium (IM'2005), Nice, France, IEEE, May 2005

5. **P. Flegkas**, P. Trimintzios, G. Pavlou, A. Liotta, *Design and Implementation of a Policy-based Resource Management Architecture*, Proceedings of IEEE/IFIP Integrated Management Symposium (IM'2003), Colorado Springs, Colorado, USA, Kluwer, pp. 215-229, March 2003

6. P. Trimintzios, T. Bauge, G. Pavlou, L. Georgiades, **P. Flegkas**, R. Egan, *Quality of Service Provisioning for Supporting Premium Services in IP Networks*, Proceedings of IEEE Globecom 2002, Taipei, Taiwan, IEEE, November 2002

7. P. Trimintzios, **P.Flegkas**, G. Pavlou, L. Georgiades, D. Griffin, *Policy-based Network Dimensioning for IP Differentiated Services Networks*, Proceedings of the IEEE Workshop on IP Operations and Management (IPOM 2002), Dallas, Texas, USA, pp. 171-176, October 2002

8. P. Trimintzios, **P.Flegkas**, G. Pavlou, *Policy-driven Traffic Engineering for Intra-domain Quality of Service Provisioning*, Proceedings of Quality of Service for future Internet Services (QofIS'02), Zurich, Switzerland, Springer, pp. 179-193, October 2002

9. **P. Flegkas**, P.Trimintzios, G. Pavlou, I. Andrikopoulos, C. F. Cavalcanti, *On Policy-based Extensible Hierarchical Network Management in QoS-enabled IP Networks*, Proceedings of

the IEEE Workshop on Policies for Distributed Systems and Networks (Policy '01), Bristol, UK, M. Sloman, J. Lobo, E. Lupu, eds., pp. 230-246, Springer-Verlag,January 2001

Book Chapters

1. O. Bonaventure, P. Trimintzios, G. Pavlou, B. Quoitin, A. Azcorra, M. Bagnulo, **P. Flegkas**, A. Garcia-Martinez, P. Georgatsos, L. Georgiadis, C. Jacquenet, L. Swinnen, S. Tandel, S. Uhlig, *Internet Traffic Engineering*, in Quality of Future Internet Services - COST Action 263 Final Report, M. Smirnov et al, eds., pp. 118-179, Springer-Verlag, 2003

Other Conference Papers

1. S. Georgoulas, G. Pavlou, **P. Flegkas**, P. Trimintzios, *Buffer and Bandwidth Management for the Expedited Forwarding Traffic Class in Differentiated Services Network*, Proceedings of the London Communications Symposium (LCS), London, UK, pp. 349-352, September 2002
2. **P. Flegkas**, P. Trimintzios, I. Andrikopoulos, G. Pavlou, *Considerations on Policy-based Network Management*, Proceedings of the London Communications Symposium (LCS), London, UK, September 2000
3. T. Baugé, R. Egan, **P. Flegkas**, P. Trimintzios, I. Andrikopoulos, G. Pavlou, *QoS Provisioning in an IP-based Studio Production Network*, Proceedings of PGNet 2000, 1st Annual PostGraduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting, Liverpool, UK, June 2000

Chapter 1

1 Introduction

With the prospect of becoming the ubiquitous, all-service network of the future, the Internet is struggling with the notion of Quality of Service (QoS). Today, the Internet Protocol (IP) architecture, which was originally conceived to offer universal connectivity with best-effort packet delivery, is being stretched by a number of proposals to improve the quality delivered to an increasingly diverse range of applications. This advent of QoS frameworks in the Internet poses new challenges in the area of network management. Management research and standardisation started in the mid-1980's where initial attempts brought mechanisms and protocols that focused on managing and configuring individual networking devices i.e. the Simple Network Management Protocol (SNMP) [SNMP]. SNMP has dominated the Internet management market, adopting the manager-agent model. This model worked well in early deployments of IP management systems for local and metropolitan area networks but now, with the evolution of Quality of Service (QoS) models, the complexity and overhead of operating and administrating networks increases enormously. As such, it is very difficult to build management systems that can cope with the growing network size, complexity and multi-service operation requirements. The emerging Policy-based Network Management paradigm claims to be a solution to these requirements.

Policy-based Management has been the subject of extensive research over the last decade [Slom94]. Policies are seen as a way to guide the behaviour of a network or distributed system through high-level, declarative directives. The Internet Engineering Task Force (IETF) has been investigating policies as a means for managing IP-based multi-service networks, focusing more on the specification of protocols and the object-oriented information models for representing policies. The declarative high-level aspect of policies is very important, particularly for human managers since it addresses the current problem of managing heterogeneous network devices in a one-by-one fashion, providing improved scalability. The second important benefit of policies is the ability of adapting the management system to changing or newly emerging requirements without having to interrupt its operation or recode part of its hardwired functionality. The aforementioned salient characteristics of policy-based management constitute the basic reasons for being widely adopted by organisations such as the IETF and Distributed Management Task Force (DMTF) as well as by many equipment vendors, e.g. CISCO, as the new promising management solution.

QoS Management has always been one of the most popular applications of policies since it enables an Internet Service Provider (ISP) to flexibly guide the behaviour of the network with respect to the different service classes offered to its customers based on its business objectives. Although a lot of work has been proposed in the literature in the area of QoS policies, most of them concentrated on defining low-level policies for configuring edge devices for the purposes of realising the Service Level Agreements (SLAs) established with the ISP customers. Existing work has thus failed to take a holistic approach towards a framework that addresses all the aspects of policy-based QoS Management of IP networks.

The objective of this thesis is to investigate the application of Policy-based Management in the context of QoS Management of IP Networks. By using policies as a means for building extensible hierarchical management systems, we propose a novel Policy-based QoS management architecture and specify the relevant policies that can drive its behaviour dynamically, providing a holistic approach to the area of policies for QoS Management. In the remaining of this chapter, we discuss the motivation behind the work presented in this thesis, highlight the research contributions and finally present an outline of the structure of the thesis.

1.1 Research Motivation

In recent years we have witnessed an evolution in network management approaches driven by the need of addressing the requirements of modern networks becoming increasingly large, sophisticated and complex. The initial protocol-based network management solutions, proposed in the early 90's as exemplified by the widely used Simple Network Management Protocol (SNMP) [SNMP], are today particularly limiting due to their centralised and static nature. In this architecture, the network elements are typically configured one-by-one, in an isolated fashion failing to cope with the growing network size, complexity and multi-service operation requirements. There is also a need to be able to program management systems as well as network devices in order to address the continuously emerging requirements of the next generation networks. This need triggered a lot of research in the area of programmable systems, with mobile code techniques considered in the context of network management aiming at building management systems with high degree of flexibility and extensibility. Despite the great advantages of mobile code approaches, they failed to be widely adopted due to the high security and safety risks they impose, given that execution of code with malicious or inadvertent bugs can seriously harm the system.

Policy-based management was seen as the solution to the above requirements and a lot of work on policies concentrated on defining high-level policy definition languages in the context of various application domains. While the high-level aspect of policies is an important benefit of policy-

based management, their programmability aspect was neglected and not addressed thoroughly in the literature. Moreover, a centralised architecture was adopted as the policy management framework where policies are executed by a centralised server in a similar manner to the SNMP management framework. The above aspects motivated the research work presented in this thesis to focus on the programmability aspects of policy-based management addressing also issues on how policies apply to hierarchically distributed management systems. We mainly view policies as a means of extending the logic of a management system at runtime, so that it can be adaptive to changing or newly emerging requirements.

With the evolution of QoS models in the Internet, policy-based management was adopted as the new promising managing paradigm by standardisation bodies such as IETF with the creation of the Policy Framework Working Group [policy-wg] and the resource allocation protocol working group [rap]. The first focused on the specification of an object oriented information model for representing policies while the second defined a framework for policy-based admission control and the specification of the Common Open Policy Service (COPS) [RFC2748] as the protocol to convey the policies from the policy server to the network nodes.

Among all the emerging QoS frameworks, the Differentiated Services (DiffServ) framework [RFC2475] is widely considered to be the most scalable approach towards QoS provisioning in the Internet. By itself, DiffServ cannot guarantee end-to-end communication properties but is limited to relative characteristics of aggregate flow behaviours at each network hop. In order to provide quantitative service guarantees, the DiffServ architecture should be augmented with intelligent operational and management functions. Although the need for a management entity was identified by the [RFC2638] where the notion of the Bandwidth Broker (BB) was proposed, a detailed architecture and specification of its functionality was never presented in the literature. In this thesis, we will present a detailed decomposition of the BB, presenting a functional architecture for managing IP DiffServ networks that provides a complete solution to QoS in the Internet. The proposed work takes the position that the future Internet should offer a *variety* of service quality levels, ranging from those with explicit, hard performance guarantees for bandwidth, loss and delay characteristics down to low-cost services based on best-effort traffic handling, with a range of services receiving qualitative traffic assurances occupying the middle ground. Service Providers will be able to negotiate Service Level Specifications (SLS) with their customers and with peer providers with confidence that the agreements can be met.

Policies are an important aspect of such a management architecture, enabling the administrator to guide the behaviour of the management system and the network according to business objectives in a flexible and scalable manner. Most of the related work in the literature has focused on policies configuring edge network devices according to the relevant SLSs but failed to address any issues related to service management and network-wide resource management. The proposed

work in this thesis attempts to provide a holistic view to QoS policies, addressing all aspects of QoS management related to areas such as SLS management and Traffic Engineering both at an offline, network-wide manner as well as at an element management level, driving dynamic operations and management functions.

1.2 Thesis Contributions

The contributions of this thesis can be grouped in two categories. The first category is related to general aspects of the policy-based management paradigm while the second category concerns its application to the area of QoS Management of IP DiffServ Networks. More specifically, the major contributions of the thesis are summarised below:

- Policies, apart from their high-level declarative nature, can be also seen as a vehicle for "late binding" functionality to management systems, allowing for their graceful evolution as requirements change. This approach of using policies to build extensible management systems differentiates it from other proposals in the literature and as such, it constitutes a research contribution in its own right. In this thesis, we are exploring the potentiality of designing "policy-aware" management systems, in which a line has to be carefully drawn between "hard-wired" functionality and policy logic. Their relationship with other approaches used to achieve programmability in systems is also discussed and we show how we can achieve programmability in a constrained manner through policies.

- We study the coexistence of policies with hierarchical management systems and propose a generic framework. The issue of high-level policy refinement into policies enforced at every layer of the management hierarchy is introduced and the cases when this methodology is applicable are identified.

- We propose a policy-based QoS architecture for managing IP Differentiated Services Networks. The proposed architecture provides a holistic approach to intra-domain QoS provisioning including both service management and traffic engineering functionality. Policies are applied to this architecture using our proposed framework for policy-based hierarchical distributed systems.

- We present a generic categorisation of QoS policies for our architecture depending on the "position" of the components they influence. The above classification differs a lot from the QoS policies identified by other researchers and the IETF which are limited to policies configuring the edge routers of a DiffServ/IntServ network. Our work provides a holistic view of QoS policies starting from SLS Management policies, network wide resource management policies down to router management policies.

- We define the policies that drive the behaviour of every component of the Policy-based QoS management architecture. Following the generic classification of QoS policies, we present SLS management and Traffic Engineering policies driving both offline as well as dynamic management components.

Other less major contributions of this thesis are also summarised below:

- We propose a generic framework for dynamically creating management services in the system through policies. The translation of policies into interpreted code, which represents the logic that combines components modelling the hardwired functionality of the system, enables the creation of services while the system is running.
- We present a formal representation of the proposed QoS policies using the object-oriented Policy Core Information Model (PCIM) and its extensions (PCIMe) as defined jointly by the IETF and DMTF standardisation bodies. We extend the core classes provided by the aforementioned models by defining specific classes and their properties needed to represent the SLS management and Traffic Engineering policies.
- Using as a case study the proposed QoS policies, we validate the methodology for applying policies to hierarchical management systems as well as the framework for management service creation using policies.
- Finally, we present the design and implementation of our prototype system in order to provide a proof of concept validation of the proposed work on policy-based QoS Management.

1.3 Thesis Roadmap

The structure of the thesis is as follows. Chapter 2 presents the background topics and the related work associated with the work of this thesis. More specifically, we first present basic network management concepts and provide an overview of the fundamental work on Policy-based Management, presenting relevant frameworks and policy representations. We then briefly describe the Differentiated Services architecture and its components and introduce the Multi-Protocol Label Switching (MPLS) technology, which is an important tool for Traffic Engineering. Finally, we present related QoS Management architectures proposed in the literature and provide an overview of the most important related work in the context of policies for QoS Management.

Chapter 3 introduces our views and considerations on building extensible management systems using policies. It then discusses the salient characteristics of hierarchical management systems and proposes a framework for the coexistence of such systems with management policies. Finally, it presents a general framework for creating management services using policies based on pre-

existing "hard-wired" functionality in the system. The proposed frameworks are validated by relevant analysis in this chapter and are also backed up by case studies in the context of QoS Management of IP Networks presented in Chapter 5.

Chapter 4 proposes a Policy-based QoS architecture for managing IP Differentiated Services networks. The proposed architecture is influenced by the TMN framework and addresses both service and network management issues, including offline and dynamic mechanisms. In this chapter, we show how policies can be applied to the proposed QoS management architecture and propose a generic classification of QoS policies that can be enforced on the various functional components of the system.

Chapter 5 defines policies that are related to QoS Management of multi-service IP networks, driving the behaviour not only of the routers of the managed network but also of the components of the management system presented in the previous chapter. We follow the generic categorisation of the QoS policies presented in chapter 4 trying to provide a holistic approach to the area of policies for QoS Management, which has not been addressed in the literature up to now. We go further and refine every category of QoS policies presented, identifying the parameters that are influenced by policies addressing both Service Management and Traffic Engineering aspects of QoS Management driving both offline and dynamic components. For the formal representation of policies we adopt an object oriented representation based on the Policy Core Information Model (PCIM) and its extensions (PCIMe) as jointly defined by the Policy Framework working group of IETF and DMTF and we propose relevant extensions to represent the proposed QoS policies. Finally, using as a case study the proposed QoS policies, we validate the methodology for applying policies to hierarchical management systems as well as the management service creation framework presented in chapter 3.

Chapter 6 presents the design and implementation of our prototype system that has been developed in order to provide a proof of concept validation of the proposed work on policies for QoS management. Specifically, we focus on the policy management sub-system components, i.e. the Policy Management Tool, the Policy Repository and the Policy Consumer, that need to be deployed over the components of the QoS management architecture presented in Chapter 4 in order to drive its behaviour dynamically through policies. We present two specific examples of the enforcement of policies applied to the offline network dimensioning (ND) and dynamic resource management (DRsM) components. We show the different way of representation of the policy rules at every stage of their life-cycle i.e. from high-level directives defined in the Policy Management Tool to an object-oriented format in the Policy Repository and finally to interpreted scripts at the Policy Consumer, presenting also their effect on the behaviour of the managed network.

Finally, Chapter 7 concludes this thesis. We summarise the contributions made and discuss their importance. This chapter closes by identifying areas in which work can be developed further and we also highlight some initial work that has been carried out in these areas.

Chapter 2

2 Background and Related Work

This chapter provides an overview of the related background topics needed for the understanding of the work presented in this thesis and provides an account of the most important related work in the area of policies for managing QoS in IP Networks. The structure of this chapter is as follows. Section 2.1 presents general network management background concepts that are used through out this thesis. Section 2.2 presents fundamental work on policy-based management, presenting relevant frameworks and policy representations that influenced the work in this thesis. Section 2.3 presents an overview of the QoS frameworks proposed by IETF and gives an overview of Multi-Protocol Label Switching (MPLS) [RFC3031] and Traffic Engineering (TE) [RFC3272]. Section 2.4 presents related QoS Management architectures proposed in the literature and Section 2.5 provides an overview of related work in the context of policies for QoS Management. Finally, Section 2.6 provides a summary of what has been presented in this chapter.

2.1 Network Management

Management research and standardisation started in the mid-1980's when the ISO/ITU-T Open Systems Interconnection Systems Management (OSI-SM) standards were specified. We present below the basic concepts defined in [X700] that have also been adopted by the Internet Management Framework and describe necessary background network management concepts for the understanding of the work presented in this thesis. The relevant introduction of the Internet Management Framework [SNMP] and the Telecommunications Management Network (TMN) Framework [M3010] is provided at the beginning of Chapter 3.

2.1.1 Management Functional Areas

OSI Systems Management standardisation followed a top-down approach, with a number of functional areas identified first. The reason for identifying those was not to describe exhaustively all relevant types of management activity but rather to investigate their key requirements and to address those through generic management infrastructure. The identified areas were *Fault*, C*onfiguration*, *Accounting*, *Performance* and *Security Management* [X700] and are collectively referred to as *FCAPS* from their initials.

Fault Management addresses the generation of error specific notifications (*alarms*), the logging of error notifications at source and the testing of network resources in order to trace and identify faults. Fault management systems should undertake alarm surveillance activities (analysis, filtering and correlation), perform resource testing and provide fault localisation and correction functions. The key requirements are event-based operation, a well-defined generic set of alarms and a testing framework.

Configuration Management deals with initialisation, installation and provisioning activities. It allows the collection of configuration and status information on demand, provides inventory facilities and supports the announcement of configuration changes through relevant notifications. The key requirements are event-based operation, control of change, generic resource state and relationship representation, scheduling, time management, software distribution and system discovery facilities.

Accounting Management deals with the collection of accounting information and its processing for charging and billing purposes. It should enable accounting limits to be set and costs to be combined when multiple resources are used in the context of a service. The key requirements are event based operation, in particular logging, and a generic usage metering framework.

Performance Management addresses the availability of information in order to determine network and system load under both natural and artificial conditions. It also supports the collection of performance information periodically in order to provide statistics and allow for capacity planning activities. Performance management needs access to a large quantity of network information and an important issue is to provide the latter with a minimum impact on the managed network. Key requirements are the ability to convert raw traffic information to traffic rates with thresholds and tidemarks applied to them; the periodic summarisation of a variety of performance information for trend identification and capacity planning; the periodic scheduling of information collection; and the ability to determine the response time between network nodes.

Security Management is concerned with two aspects of systems security. The *management of security*, which requires the ability to monitor and control the availability of security facilities and to report security threats or breaches. And the *security of management,* which requires the ability to authenticate management users and applications, to guarantee the confidentiality and integrity of management exchanges and to prevent unauthorised access to management information.

2.1.2 The Manager-Agent model

OSI management has introduced the manager-agent model. A simplified version of this model has also been adopted by the Internet SNMP [SNMP]. According to the model, manageable resources are modelled by managed objects at different levels of abstraction. Managed objects encapsulate

the underlying resource and offer an abstract access interface at the object boundary. The management aspects of entities such as network elements and distributed applications are modelled through "clusters" of managed objects, seen collectively across a management interface. The latter is defined through the formal specification of the relevant managed object types or classes and the associated access mechanism, i.e. the management access service and supporting protocol stack. Management interfaces can be thought as "exported" by applications in agent roles and "imported" by applications in manager roles. Manager applications access managed objects across interfaces in order to implement management policies.

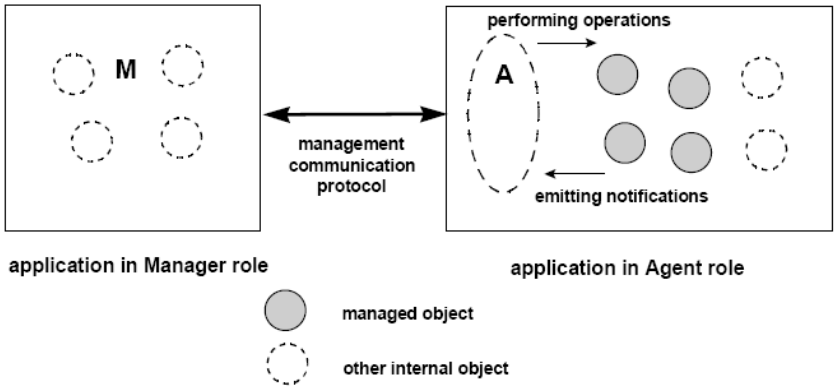


Figure 2-1 The Manager-Agent model

OSI and Internet management is primarily a communications framework. Standardisation affects the way in which management information is modelled and carried across systems, leaving deliberately unspecified aspects of their internal structure. The manager-agent model is shown in Figure 2-1. Note that manager and agent applications contain other internal objects that support the implementation of relevant functionality. These are not visible externally, so they are depicted with dotted lines.

The management access service and protocol carries the parameters of operations to managed objects and returns relevant results, so it can be loosely described as a "remote method execution" protocol (in object-oriented systems, an object's procedure is called a "method"). The relevant parameters and results are a superset of those available at the object boundary, allowing to dereference an object by name and to select a number of objects to perform an operation. In fact, the agent offers an object-oriented database-like facility which has the effect that *one* operation across the network may result in *many* operations to managed objects inside the agent, with a "consolidated" result passed back. In the other direction, notifications emitted by managed objects are discriminated internally within the agent, based on criteria preset by manager applications. This mechanism eliminates unwanted notifications at source and forwards useful notifications *directly* to interested managers.

2.2 Policy-based Management

In this section, we present the two most important and widely accepted works on policy-based management. First, we present the work on policies that has been carried out at Imperial College London and has pioneered most of the concepts on policy management. We continue by presenting the work on policy-based network management carried out in the IETF by two working groups. These works have influenced most of the related work in the literature as well as the work presented in this thesis.

2.2.1 The Ponder policy framework

[Slom94] presents the concepts of domains and policies in the context of a generic management architecture. Assuming a distributed management system that reflects the distribution of the system being managed, policies are specified as objects, which define a relationship between *subjects* (managers) and *targets* (managed objects). Policies are separated from the automated managers, facilitating the dynamic change of the behaviour and the adaptivity to new requirements without re-implementing the management applications. Domains provide the framework for partitioning management responsibilities by grouping objects in order to specify a management policy that applies to a domain. Domains are defined as objects, which maintain a list of references to their member managed objects [Slom89]. This concept of domain-based policy is adopted by many researchers in the area of policy-based management [Alpers95] [Wies95] [Putt95] [Schw94] [Koch95].

[Slom99] gives a general description of the ponder policy framework depicted in Figure 2-2. An administrator creates and modifies policies using the Policy Editor. Authorization policies are disseminated to target agents as specified by the target domains and obligation policies to manager-agent applications as specified by the subject domains. Manager agents interpret policies, which can be enabled, disabled or removed from the application and register with the monitoring service to receive events that trigger one or more policies. On receiving an event, the manager-agent queries the domain service to determine the target objects, and performs the policy actions on them. Detailed description of the components of this architecture as well as the interactions between them is provided by [Marr97].

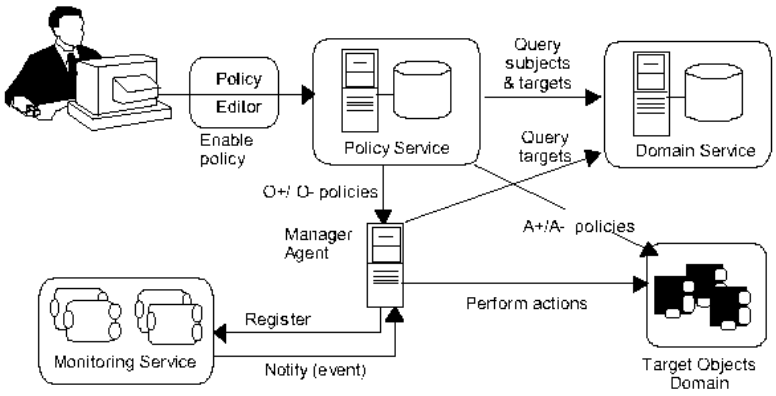


Figure 2-2 The Ponder Policy Management Architecture

[Dami02] has progressed the initial work on the ponder framework and presented a deployment model that is object-oriented and addresses the instantiation, distribution and enabling of policies as well as the disabling, unloading and deletion of policies. The model defines objects for policies, for domains and for policy enforcement agents and outlines the interactions needed between them. In the following section, we give an overview of the ponder policy specification language.

2.2.1.1 The Ponder policy specification language

Ponder is a declarative, object-oriented language [Dami01] for specifying security policies with role-based access control, as well as general-purpose management policies for specifying what actions are carried out when specific events occur within the system or what resources to allocate under specific conditions. Ponder has four basic policy types: authorisations, obligations, refrains and delegations.

Authorisations and Delegations

Authorisation policies are designed to protect target objects and are conceptually enforced by the target objects. In practice, authorisation policy enforcement is delegated to one or more enforcement agents that intercept actions and perform checks on whether the access is permitted. The enforcement agents for authorisation policies are termed access controllers and typically interface to lower-level access control mechanisms that really carry out the access control, for example a firewall protecting the services on its network, an operating system protecting its resources, or a database manager protecting its databases. An access controller will normally protect all the targets at its location and enforce all authorisation policies relating to them.

Ponder supports both positive authorisations that permit an action, and negative authorisations that forbid an action. Authorisations can be constrained by Boolean expressions and essentially handle the common functionality found in existing access control mechanisms. The following

example gives a flavour of the language. The policy states that all members of the secretaries domain are permitted to send documents for printing to spoolers in the colour printers domain, but only between 0900 and 1700 and only if the document to print is no longer than 10 pages.

```
type auth+ printing (subject S, target T, int validfrom, int validto, int maxPages) {
action T.print (document);
when time.between (validfrom, validto) && document.size () <= maxPages;
}
inst auth+ printingpolicy = printing ( /secretaries, /printers/colour, 0900, 1700, 10);
```

For delegation, Ponder takes a simple approach whereby a delegation policy can be written to permit the subjects of an authorisation policy (grantors) to delegate some or all of their access rights to a new set of subjects (grantees). Effectively, when a grantor performs a delegation action, a new authorisation policy is created. Since this new authorisation policy is identical to the original authorisation policy except for a new subject set, the implementation and enforcement of delegated policies is the same as that for authorisation policies and is not considered further in this paper. Note that delegation does not transfer access rights from a grantor to the grantee set; grantors retain their access rights after a delegation is performed. Ponder also supports constraints on delegation policies, negative delegation policies and cascaded delegation.

Refrains and Obligations

While the subjects of an authorisation policy can be any objects that initiate invocations, the subjects of refrain and obligation policies are instances of special enforcement agents called policy management agents (PMAs), whose behaviour is defined by the refrain and obligation policies that apply to them (or the real-world entity that they represent). Policy management agents thus enforce all the refrain and obligation policies for a subject directly. Such agents will normally be generic, although multiple implementations are allowed since any object that implements a PMA interface can be the subject of refrain and obligation policies.

Refrains define what actions a subject is not permitted to invoke. Refrains are similar to negative authorisations but are enforced at the subject by the policy management agent and apply to the actions the subject invokes. Refrains are used where we do not trust the targets to enforce a policy. In contrast to the other basic policy types, which are essentially access control policies, obligations are event-triggered policies that carry out management tasks on a set of target objects or on the subject itself. Obligation policies allow us to automate systems, for example, when security violations occur; when resources need to be reconfigured in response to quality-of-service degradation, etc. In the following example, when a print error event occurs, the

printManager policy management agent will notify all operators of the error and log the event internally (the -> symbol below means sequential execution of the actions).

```
type oblig+ printManagement (subject S, target T) {
on printError (printer, error);
do T.notify (printer, error) -> S.log (printer, error);
}
inst oblig p2 = printManagement (/printManager, /operators)
```

2.2.2 The IEFT Policy Management Frameworks

Two working groups in the IETF have considered policy management or *policy-based networking*: the Resource Allocation Protocol (RAP) Working Group (WG) [rap-wg] and the Policy Framework WG [policy-wg]. The purpose of the RAP WG is to establish a scalable policy control model for RSVP and specify a protocol for use among RSVP-capable network nodes and policy servers. The Policy WG has provided several documents describing a general framework for representing, managing, sharing and reusing policies in a vendor independent, interoperable and scalable manner as well as defining an extensible information model for representing policies and an extension to this model to address the need for QoS management.

The RAP WG has described a framework for policy-based admission control specifying the two main architectural elements as shown in Figure 2-3 (a) [RFC2753]: the Policy Enforcement Point (PEP) and the Policy Decision Point (PDP). PEP represents the component that always runs on the policy-aware node and it is the point where the policy decisions are actually enforced. The PDP is the point where the policy decisions are made. When a PEP receives a notification or a message that requires a policy decision, it creates a request that includes information which describes the admission control request. Then, the PEP may consult a local configuration database to identify which policy elements can be evaluated locally, passes the request with this set to the Local Policy Decision Point (LPDP) and receives the result. The PEP then passes all the policy elements and the partial result to the PDP, which combines its result with the partial result from the LPDP and returns the final policy decision to the PEP.

The RAP WG has also specified a protocol that allows policy servers (PDPs) to communicate policy decisions to network devices (PEPs) called the Common Open Policy Service (COPS) [RFC2748]. COPS is a query/response protocol that supports two common models for policy control: Outsourcing and Configuration.

The Outsourcing model addresses the type of events at the PEP that require an instantaneous policy decision (authorization). In the outsourcing scenario, the PEP delegates responsibility to

an external policy server (PDP) to make decisions on its behalf. For example, in COPS Usage for RSVP [COPSRSVP] when a RSVP reservation message arrives, the PEP must decide whether to admit or reject the request. It can outsource this decision by sending a specific query to its PDP, waiting for its decision before admitting the outstanding reservation.

The COPS Configuration model [COPS-PR], on the other hand, makes no assumptions of such direct 1:1 correlation between PEP events and PDP decisions. The PDP may proactively provision the PEP reacting to external events (such as user input), PEP events, and any combination. Provisioning may be performed in bulk (e.g., entire router QoS configuration) or in portions (e.g., updating a DiffServ marking filter).

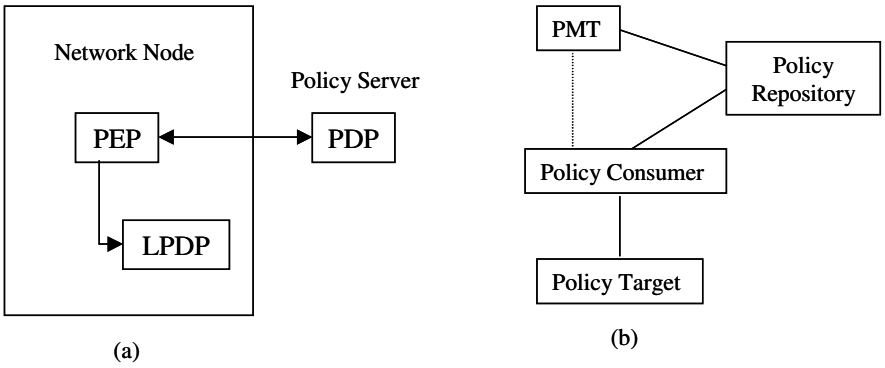


Figure 2-3 a) RAP WG Policy Framework for Admission Control, b) Policy WG Framework.

The Policy Framework WG defines policy as an aggregation of *Policy Rules*. Each policy rule comprises a set of conditions and a corresponding set of actions that are intended to be device- and vendor-independent. Policy Rules are of the form: if <condition> then < action>. The <condition> expression may be a compound expression and it may be related to entities such as hosts, applications, protocols, users, etc. The <action> expression may be a set of actions that specify services to grant or deny or other parameters to be input to the provision of one or more services.

The four major functional elements of the Policy Framework (Figure 2-3b) described by this group are:

- A **Policy Management Tool** to enable an entity to define, update and optionally monitor the deployment of Policy Rules.
- A **Policy Repository** to store and retrieve Policy Rules.
- A **Policy Consumer** which is a convenient grouping of functions, responsible for acquiring, deploying and optionally translating Policy Rules into a form useable for Policy Targets.

- A **Policy Target,** which is an element whose behaviour is dictated by Policy Rules carrying out the action indicated by the Policy Rule.

A detailed description of the functionality of each element can be found in [Stev99].

2.2.2.1 The Policy Core Information model (PCIM) and its Extensions (PCIMe)

The DMTF [DMTF], in collaboration with the IETF Policy work group [policy-wg], defined a policy information model as an extension to the Common Information Model (CIM) [CIM], known as PCIM [RFC3060]. In this section, we will describe in detail PCIM since the work presented in this thesis extends PCIM to represent the proposed QoS policies. An information model is "an abstraction and representation of the entities in a managed environment — their properties, operation, and relationships." This is independent of any specific repository, application, protocol, or platform. The IETF defined a mapping of the PCIM model to a directory schema so that the Lightweight Directory Access Protocol (LDAP) directory can be used as a repository [RFC3703]. The CIM defines generic objects such as managed system elements, logical and physical elements, systems, service, and service access point. The Policy Model defines a policy rule and its component policy conditions and policy actions as shown in Figure 2-4. The assumption is that a policy rule is of the form `if <condition set> then do <action list>`. The condition set can be expressed in either disjunctive (DNF) or conjunctive normal form (CNF).

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

In Figure 2-4, the boxes represent the classes, and the dotted arrows represent the associations. There appear the following associations: *PolicyGroupInPolicyGroup*, *PolicyGroupInSystem*, *PolicyRuleInSystem*, *PolicyRepositoryInPolicyRepository*, P*olicyRuleInPolicyGroup*, *Policy-ConditionInPolicyRepository*, P*olicyActionInPolicyRepository*, *PolicyConditionInPolicyRule*, *PolicyRuleValidityPeriod* and *PolicyActionInPolicyRule*. An association always connects two classes. These classes may, however, be the same class, as is the case with the *PolicyGroupInPolicyGroup* association, which represents the recursive containment of *PolicyGroups* in other *PolicyGroups*. The *PolicyRepositoryInPolicyRepository* association is recursive in the same way. Cardinalities indicate how many instances of each class may be related to an instance of the other class and they are included into associations.

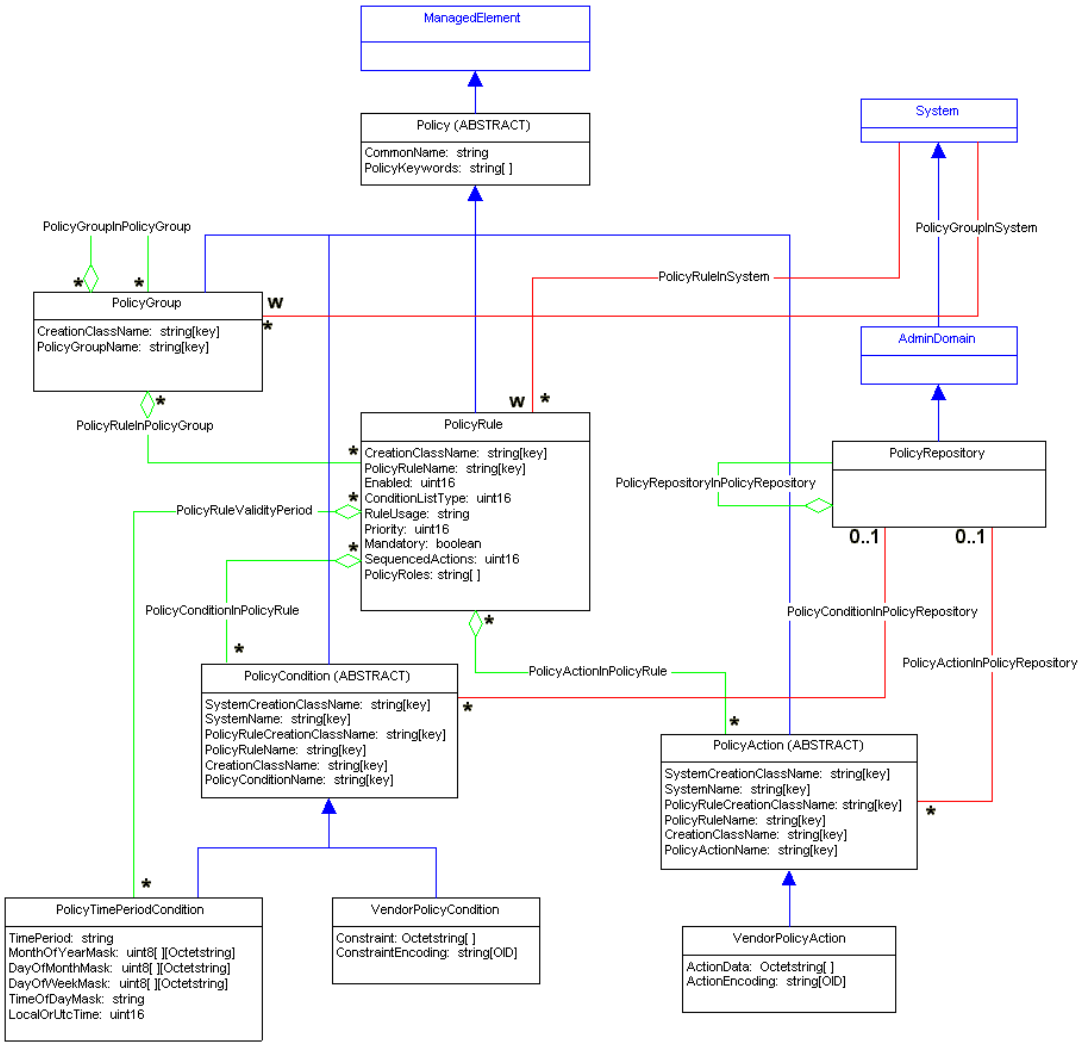


Figure 2-4 The Policy Core Information Model

After PCIM became a standard track RFC, the IETF Policy Framework WG produced a document that proposed a number of changes to the Policy Core Information Model (PCIM) [RFC3460]. Two types of changes are included. First, several completely new elements are introduced, for example, classes for header filtering, that extend PCIM into areas that it did not previously cover. Second, there are cases where elements of PCIM (for example, policy rule priorities) are deprecated, and replacement elements are defined (in this case, priorities tied to associations that refer to policy rules). Both types of changes are done in such a way that, to the extent possible, interoperability with implementations of the original PCIM model is preserved.

We discuss below two of the most important changes that are useful for the understanding of the work presented in this thesis. A detailed description of the changes can be found in [RFC3460].

The concept of a CompoundPolicyCondition and a CompoundPolicyAction has been introduced in PCIMe. In both cases the idea is to create reusable "chunks" of policy that can exist as named

elements in a ReusablePolicyContainer (Deprecated PolicyRepository). The "Compound" classes and their associations incorporate the condition and action semantics that PCIM defined at the PolicyRule level: DNF/CNF for conditions, and ordering for actions. Compound conditions and actions are defined to work with any component conditions and actions. In other words, the components may be instances, respectively, of SimplePolicyCondition and SimplePolicyAction (discussed immediately below) or conditions and actions directly derived from the classes PolicyCondition and PolicyAction.

The SimplePolicyCondition / PolicyVariable / PolicyValue structure has been introduced into PCIMe. A list of PCIMe-level variables is defined, as well as a list of PCIMe-level values. Other variables and values may, if necessary, be defined in submodels of PCIMe. A corresponding SimplePolicyAction / PolicyVariable / PolicyValue structure is also defined. While the semantics of a SimplePolicyCondition are "variable matches value", a SimplePolicyAction has the semantics "set variable to value".

2.3 QoS in IP Networks

The Internet is currently built around the Best-Effort (BE) service model. This model has been considered adequate for a long time, its simplicity and efficiency contributing towards the widespread deployment of the Internet. On the other hand, this continuous growth and the achieved ubiquity have resulted in increasing demand for new services, which require more sophisticated service models. With the prospect of becoming the ubiquitous all-service network of the future, the Internet needs to evolve to support services with guaranteed Quality of Service (QoS) characteristics. This has prompted the research community to devise a number of approaches for providing QoS to Internet applications. As a result, the Internet Engineering Task Force (IETF) has proposed in recent years two QoS models, the Integrated [RFC1633] and Differentiated Services [RFC2475].

In the following sections, a short overview of the IntServ model is given while the DiffServ model is described in more detail.

2.3.1 Overview of Integrated Services

The Integrated Services (IntServ) model follows an approach similar to that found in multi-service telecommunication networks, most notably in Asynchronous Transfer Mode (ATM) networks. In this model there is a hard sense of QoS in terms of resources allocated to individual flows, with the Resource Reservation Protocol (RSVP) [RFC2209] used for signalling the required QoS characteristics to the network. With flow state information required at every router in the path between receiver and transmitter, scalability has been the main architectural concern and one of

the main reasons that restricted its deployment. The amount of state information increases proportionally with the number of flows, thus placing a huge storage and processing overhead to the routers and requiring fairly complex control components in each router.

The Integrated Services model supports two new classes of service in addition to the existing best-effort class. These are the Guaranteed and the Controlled Load service classes. The Guaranteed Service class [RFC2212] is a quantitative service, which provides strong guarantees in terms of end-to-end delay and bandwidth but it does not attempt to guarantee jitter. It also ensures that no packet will be discarded due to queues overflowing anywhere in the network. The Controlled Load service [RFC2211] is a qualitative service and it is defined as being equivalent to the service obtained using best effort on a lightly loaded network. If the load on the network increases the best effort traffic will find its service quality degraded while Controlled Load traffic will still receive the service it got under the light load scenario.

As mentioned previously, the reservation mechanism used in IntServ is the RSVP signalling protocol. RSVP is a simplex, receiver-oriented soft-state[1] protocol. When a reservation is required from A to B across a network, A sends an RSVP PATH message containing a description of the flow. This message is routed across the network using the underlying routing protocol. At each router along the way, the local IntServ entity makes a note of the previous hop, updates parameters in its memory and amends some of the IntServ objects carried by the message. Once the PATH message reaches its destination B, it contains end to end information about the routers capabilities (as specific objects were updated along the way). The receiver may then initiate a reservation request, based on the PATH information received. It specifies the characteristics of the reservation (service type, resource requirements…) and composes a RESV message. This message is routed back to A using the previous hop information stored at each router (thus ensuring that the same path is used in both journeys). If a node along the way is unable to accommodate the reservation, an error is generated and the RESV message is not forwarded any further. If all intermediate routers can accommodate it, an end-to-end reservation is set up. A confirmation of success may be returned if the destination has requested it.

2.3.2 Differentiated Services

Differentiated services, as proposed by the IETF Differentiated Services Working Group [diffserv-wg], allow IP traffic to be classified into a finite number of service classes that receive different router treatment. For example, traffic belonging to a higher priority and/or delay service class receives some form of preferential treatment over traffic classified into a lower service class.

[1] RSVP sends periodic refresh messages to maintain the state along the established path(s).

Differentiated services do not attempt to give explicit end-to-end guarantees. Instead, in congested network elements, traffic with a higher class of priority has a higher probability of getting through, or in case of delay priority, is scheduled for transmission before traffic that is more delay-tolerant.

The DiffServ follows a "keep all complexity at the network edges" approach where all complicated per-flow packet processing is done at the network boundaries. The information required to perform actual differentiation in the network elements is carried in the Type of Service (ToS) field of the IPv4 packet headers or the Traffic Class field of the IPv6 packet headers, referred to as the DiffServ Field or Codepoint (DSCP) [RFC2474]. Thus, since the information required by the buffer management and scheduling mechanisms is carried within the packet, differentiated services do not require signalling protocols to control the mechanisms that are used to select different treatment for the individual packets. Consequently, the amount of state information, which is required to be maintained per node, is proportional to the number of service classes and not proportional to the number of application flows.

At each differentiated services user/provider boundary, the service provided is defined by means of a Service Level Agreement (SLA). The SLA is a contract, established either statically or dynamically, that specifies the overall performance and features, which can be expected by a customer. Because differentiated services are for unidirectional traffic only, each direction must be considered separately. The subset of the SLA, which provides the technical specification of the service, is referred to as the Service Level Specification (SLS).

An important subset of the SLS is the Traffic Conditioning Specification (TCS), which specifies detailed service parameters for each service level. These service parameters include service performance parameters (e.g. throughput, latency, drop probability) and traffic profiles corresponding to the requested service. Furthermore, the TCS may define the marking and shaping functions to be provided.

In the next section, the major functional elements of the DiffServ architecture are described in more detail.

2.3.2.1 Functional Elements of the Differentiated Services Architecture

The Differentiated Services architecture is composed of a number of functional elements, namely packet classifiers, traffic conditioners and per-hop forwarding behaviours (PHBs). According to the basic differentiated services architecture definition, these elements are normally placed in ingress and egress boundary nodes of a differentiated services domain and in interior DiffServ-compliant nodes (Figure 2-5). However, it is not necessary for all the elements to be present in all

the DiffServ-compliant nodes, something that strictly depends on the functionality that is required at each node [RFC2475].

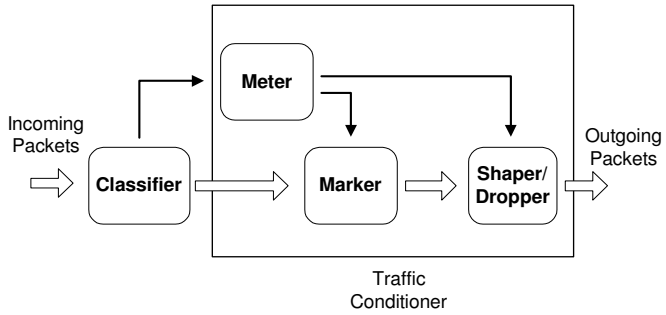


Figure 2-5 Typical arrangement of a Packet Classifier and a Traffic Conditioner.

Packet classification is a significant function, which is normally required at the edge of the differentiated services network. Its goal is to provide identification of the packets belonging to a traffic stream that may receive differentiated services. Classification is done with packet classifiers, which select packets based on the content of packet headers according to well-defined rules determined by the TCS. Two types of classifiers are currently defined: the Behaviour Aggregate (BA) classifier, which selects packets based on the DiffServ Codepoint only, and the Multi-Field (MF) classifier, which performs the selection based on the combination of one or more header fields.

Traffic conditioners form the most vital part of a differentiated services network. Their goal is to apply conditioning functions on the previously classified packets according to a predefined TCS. A traffic conditioner consists of one or more of the following components: a) a meter device, which measures the temporal properties of a traffic stream selected by a classifier b) a marker device that sets the DiffServ Codepoint in a packet based on well-defined rules c) a shaper device that delays packets within a traffic stream to cause the stream to conform to some defined traffic profile and d) a dropper/policer device that discards packets based on specified rules (e.g. when the traffic stream does not conform to its TCS).

A *Per-Hop Forwarding Behaviour (PHB)* is a description of the externally observable forwarding behaviour of a differentiated services node, applied to a collection of packets with the same DSCP that are crossing a link in a particular direction (called differentiated services behaviour aggregate). Each service class is associated with a PHB. DSCP marking will typically be performed only once at the network boundary, thereby marking each packet for a specific PHB according to a pre-arranged service level specification. Currently, there are three proposed PHBs: a) *The Default (DE) PHB* which is the common, best-effort forwarding available in today's Internet, b) the Expedited Forwarding (EF) PHB that is a high priority behaviour, defined as a forwarding treatment for a particular differentiated services aggregate where the departure rate of

the aggregate's packets from any DiffServ-compliant node must equal or exceed a configurable rate [RFC2598] and c) the Assured Forwarding (AF) PHB that is a means for a provider differentiated services domain to offer different levels of forwarding assurances for IP packets received from a customer differentiated services domain. Four AF classes are defined, where each AF class in each differentiated services node is allocated a certain amount of forwarding resources, e.g. buffer space and bandwidth. Within each AF class, IP packets are marked with one of three possible drop precedence values. In case of congestion, the drop precedence of a packet determines the relative importance of the packet within the AF class [RFC2597].

The Diffserv WG [diffserv-wg], as described above, has defined the general architecture for differentiated services and has focused on the forwarding path behaviour required at the level of each router. The WG also recognised that this is not enough for offering a QoS solution, so it went a step further to define the notion of the Per-Domain Behaviour (PDB). According to [RFC3086] a PDB is the expected treatment that an identifiable group of packets will receive from edge-to-edge in a DS domain. A particular PHB (or, if applicable, list of PHBs) and traffic conditioning requirements are associated with each PDB. The WG did set the rules for describing PDBs but did not produce any output specifying a PDB, other than the default PDB, which is based on the Default PHB and is nothing more than the common best-effort behaviour of current IP networks.

2.3.3 Multi-Protocol Label Switching and Traffic Engineering

Multi-Protocol Label Switching (MPLS) is a forwarding scheme [RFC3031]. The motivation for the introduction of MPLS was the seamless support of IP over the Asynchronous Transfer Mode (ATM) switching infrastructure. The technique used by MPLS for forwarding is called label-switching. A short fixed length label is associated with each segment of a path in a domain that supports MPLS. These paths are called Label Switched Paths (LSPs), while the routers that support MPLS are called Label-Switched Routers (LSRs). LSPs are configured with a label distribution protocol. The routing of LSPs can either be based on the hop-by-hop IP routing decisions, or can be based on other explicit routing processes.

MPLS is considered an important emerging technology for enhancing IP in both features and services. Although, the concept of Traffic Engineering does not depend on specific layer 2 technologies, it is argued that MPLS is the most suitable tool to provide it. MPLS allows sophisticated routing control capabilities as well as QoS resource management techniques to be introduced to IP networks [RFC3564]. With the advent of Differentiated Services and MPLS, IP traffic engineering has attracted a lot of attention in recent years. [Auki00], [Feld00], [QBONE], are a few of the most recent projects in this area.

Traffic Engineering (TE) [RFC3272] is in general the process of specifying the manner in which traffic is treated within a given network. TE has both user and system-oriented objectives. The users expect certain performance from the network, which in turn should attempt to satisfy these expectations. The expected performance depends on the type of traffic that the network carries, and is specified in the SLS contract between customer and Internet Service Provider (ISP). The network operator on the other hand should attempt to satisfy the user traffic requirements in a cost-effective manner. Hence, the target is to accommodate as many as possible of the traffic requests by using optimally the available network resources. Both objectives are difficult to realise in a multi-service network environment.

2.4 Related work on QoS Management

QoS frameworks such as IntServ and DiffServ have so far concentrated in control plane mechanisms for providing QoS. However, it would not seem possible to provide QoS without the network and service management support, which is an integral part of QoS-based telecommunication networks. Considering in particular the DiffServ architecture, a key issue is end-to-end QoS delivery. The DiffServ architecture suggests only mechanisms for relative packet forwarding treatment to aggregate flows, traffic management and conditioning; by no means does it suggest an architecture for end-to-end QoS delivery. In order to provide end-to-end quantitative QoS guarantees, DiffServ mechanisms should be augmented with intelligent traffic engineering functions. The work in [RFC2638] was the first, which proposed the notion of the centralized management entity called the *Bandwidth Broker (BB)* shown in Figure 2-6, as the means to support the *Premium Service.* This work was a high level presentation of the required components such as QoS provisioning, admission control etc for DiffServ networks. BBs can be configured with organizational policies, keep track of the current allocation of marked traffic, and interpret new requests to mark traffic in light of the policies and current allocation.

In short, a bandwidth broker manages network resources for IP QoS services supported in the network based on the Service Level Specifications (SLS) and used by customers of the network services. A BB may be considered a type of policy manager in that it performs a subset of policy management functionality. Ideally, bandwidth brokers and policy managers will work together to provide call admission control in an integrated policy services environment.

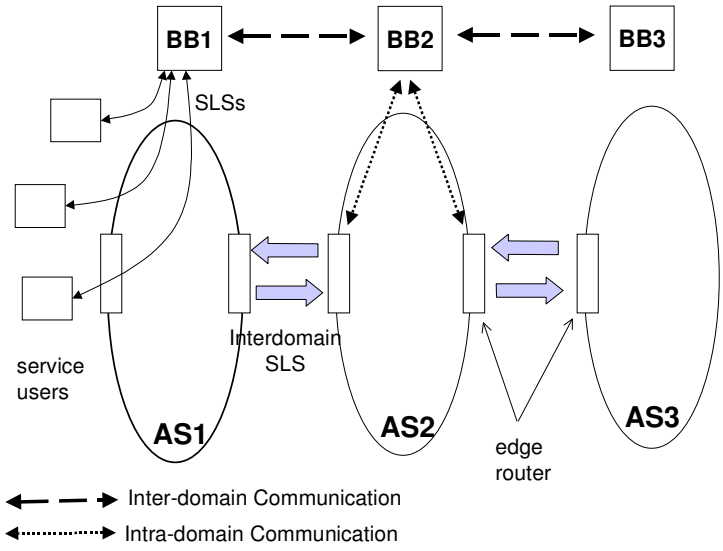


Figure 2-6 The use of Bandwidth Brokers to manage diverse network domains

The BB gathers and monitors the state of QoS resources within its domain and on the edges of the domain going to and from adjacent domains. That information, together with the policy information (from the policy rules data base) is used for admission control decisions on QoS service requests to the network. The network state information from the BB is also used to verify that resources are currently available in the network to support the request. This information may be obtained directly from the BB via an interface to the policy manager or, more likely, the BB places it in a common database shared by the policy manager.

BBs have two responsibilities. Their primary one is to parcel out their region's marked traffic allocations and set up the leaf routers within the local domain. The other is to manage the messages that are sent across boundaries to adjacent regions' BBs. A BB is associated with a particular trust region, one per domain. Only a BB can configure the leaf routers to deliver a particular service to flows, crucial for deploying a secure system. The BB is responsible for managing inter-domain communication, with BBs in neighbouring networks, for the purpose of coordinating.

When an allocation is desired for a particular flow, a request is sent to the BB. The request can be made manually by a network administrator or a user or it might come from another region's BB. A BB first authenticates the credentials of the requester, and then verifies that there exists unallocated bandwidth sufficient to meet the request. If a request passes these tests, the available bandwidth is reduced by the requested amount and the flow specification is recorded. In the case where the flow has a destination outside this trust region, the request must fall within the class allocation through the "next hop" trust region that was established through a bilateral agreement

of the two trust regions. The requester's BB informs the adjacent region's BB that it will be using some of this rate allocation. The BB configures the appropriate leaf router with the information about the packet flow to be given a service at the time that the service is to commence. This configuration is "soft state" that the BB will periodically refresh. The BB in the adjacent region is responsible for configuring the border router to permit the allocated packet flow to pass and for any additional configurations and negotiations within and across its borders that will allow the flow to reach its final destination A lot of work has been done along the research community defining and implementing BB architectures such as [Neil99], [Terz99], [Hoo99], [BCIT].

In [Feld01] the authors presented the problem of intra-domain provisioning for the purpose of balancing the load on the network. This work suggested a centralized tool, which aims to automatically calculate the routing configuration based on a traffic matrix calculated from measurements. In [Zhan00] the authors present a bandwidth broker architecture in centralized manner, where the network nodes do not have to keep any QoS state information. The paper concentrates on the admission control based on the virtual time reference system proposed by the authors. The work in [Auki00] presented a routing and traffic engineering server in order to enforce the authors' suggested minimum interference routing. To the best of our knowledge, none of the related works considers both service-related aspects and traffic engineering, and in addition none of them considers multiple levels in traffic engineering or admission control.

2.5 Related Work on QoS Policies

[Verm01a] presents a policy-based management framework for managing SLAs in an IP DiffServ network of an enterprise environment. A scheme is proposed that enables a network administrator to manage and configure DiffServ networks from a central location and also abstract away the specific details of device configuration, and allow him/her to express the management of the network in terms of application-oriented performance metrics.

It has also proposed a QoS Management tool architecture [Verm01b] used by a network administrator to configure and administer the DiffServ components in an enterprise network. The components of the management tool are as shown in Figure 2-7.

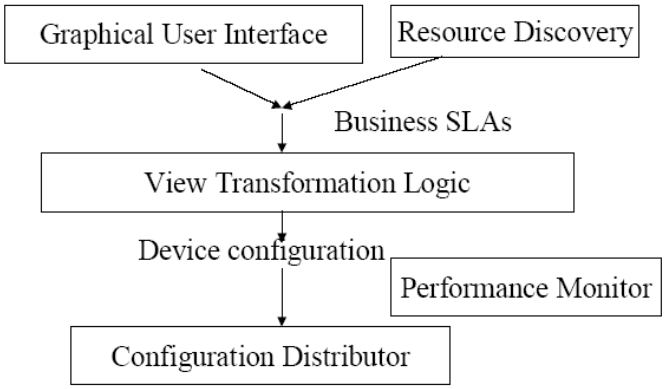


Figure 2-7 Components of the QoS Management Tool

The QoS management tool consists of five major components:

- The *Graphical User Interface* is the means by which an administrator can input the objectives for deploying QoS in the network. The objectives constitute the business Service Level Agreements (SLAs) that the network is required to meet to satisfy the performance needs to its customers.
- The *Resource Discovery* component is responsible for determining the topology of the network and the users and applications that are operational in the network. The component is also responsible for identifying the capabilities of each device in the network, e.g. the set of PHBs that are supported at the core routers, or the capabilities of a server to retrieve configuration information stored in the network.
- The *view transformation logic* component is responsible for ensuring that the business SLAs that are specified by the network administrator are mutually consistent, correct and feasible with the existing capacity and topology of the network. It also translates the business SLAs into device configuration information that can be distributed to the different components.
- The *Configuration Distributor* is responsible for ensuring the device configuration is distributed to the various devices in the network. The distribution of the configuration can be done in a variety of ways. One way would be by storing them into a repository from where different devices can retrieve it; making a configuration file and copying it over to the device. Another way is to have a program, which can log into the device console remotely and issue commands to configure it appropriately.
- The *Performance monitor* keeps track of the performance of the network, and compares the performance obtained by different traffic flows in the network. It determines whether the business SLAs specified by the network administrator are being satisfied or not.

Both the business SLAs and the device configuration are specified in XML using a specialized Document Type Definition (DTD). These SLAs are defined in terms of the application response time that occurs when a specific client accesses an application. The user-defined high-level policies will map each usage scenario, a client accessing an application on a server, to a service class. The service class has specific performance objectives associated with it. A set of six tables is used in this framework, described in detail in [Verm01a], to represent these policies, as shown in Figure 2.10. A table of users provides the mapping of users to the different subnets and IP addresses. A table of applications provides information about the port numbers that applications will use. A table of routers and a table of servers provide information about the different policy enforcement points that exist in the network, and whose configuration needs to be generated. The fifth table provides information about the different service levels that are defined within the network. The entries in the tables of users, applications, servers, routers and service classes are tied together with entries in the table of policies, which maps the different application flows to different classes of service.

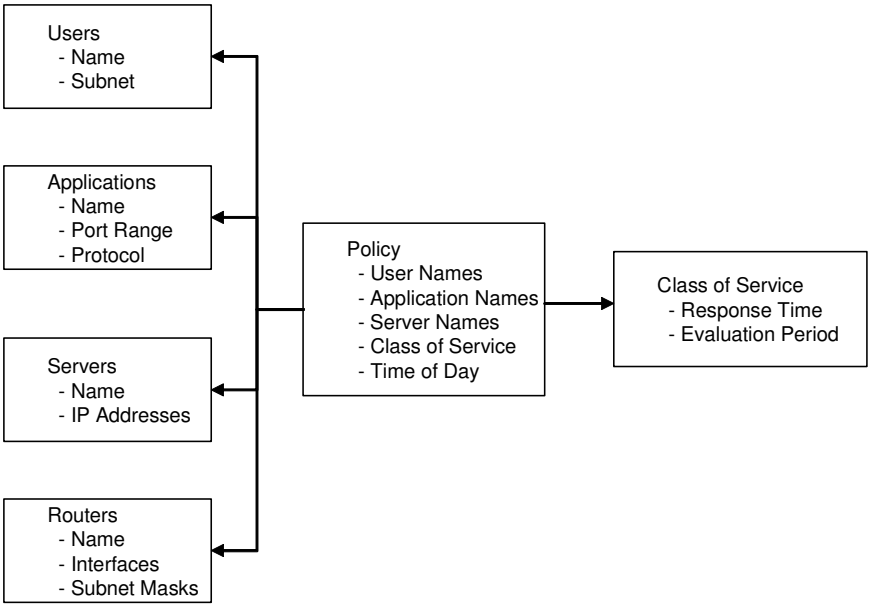


Figure 2-8 High-level SLA Policy Representation

After the high-level policies mentioned above are defined, and the relevant checks are performed, the final step is to translate these high-level policies to low-level ones that correspond to the configuration of each device that is needed to support the business need. For performing this low-level translation, they defined a representation of the low-level policy that would be required at each policy enforcement point (or policy target). For each device within the network, two tables are defined – one defining a set of classification policies and the other defining the different network levels supported at the device. A classification policy contains the five tuples that describe an IP packet flow (source and destination address, source and destination port and

protocol) and a mapping to the network level. The network-level definition contains DiffServ-specific details, such as the appropriate marking within the IP header, and the rates that are appropriate for each of the network levels. The translation process maps the representation specified in terms of Figure 2-8 to the representation in terms of Figure 2-9.

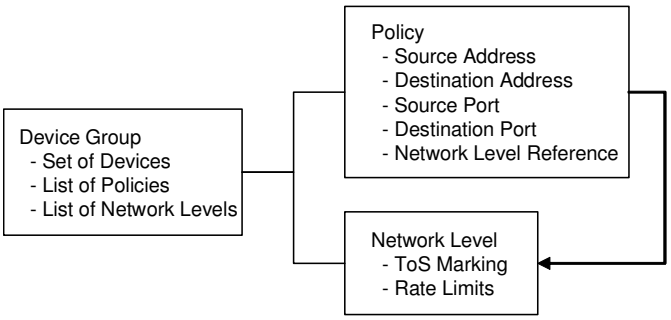


Figure 2-9 Low-level policy for device configuration

This translation process consists of four steps: name mapping, in which the high-level constructs like names of applications or users are mapped to header fields like IP addresses and port numbers; Class of Service (CoS) mapping, in which the definition of high-level classes of service (e.g. Precedence) is changed into technology-specific classes of service (e.g. EF PHB for DiffServ); PHB's relevance determination, where the relevancy of a policy to a device or a set of devices is examined (i.e. to determine the set of devices that are affected by each of the defined policies); and grouping, wherein devices which have identical sets of applicable policies are grouped together in a common structure.

The work described above [Verma01] defines only policies for realising SLAs by configuring the edge routers i.e. mapping flows to QoS classes supported in the DiffServ network. These policies are static and do not depend on any network state. The flexibility of changing dynamically the configuration based on events and conditions of the system or the network through policies is not addressed. Moreover, policies that define the network-wide behaviour are not considered, achieving only a high-level configuration of the network nodes. Although this work considers SLA related policies, these are related to how SLAs are translated to low-level device configuration and not how to dynamically drive the behaviour of the QoS Management system and the network. Finally, the issues of managing SLAs such as admission control and negotiation as well as resource management have not been studied.

The next important work related to QoS policies is the one presented by the IETF Policy Framework Working group. [RFC3644] presents an object-oriented information model for representing Quality of Service network management policies. The QoS Policy Information Model (QPIM) is based on the IETF Policy Core Information Model and its extensions and defines an information model for QoS enforcement for differentiated and integrated services using

policies. QPIM provides actions and conditions that control the classification, policing and shaping done within the differentiated service domain boundaries, as well as actions that control the per-hop behavior within the core of the DiffServ network. QPIM does not mandate the use of DiffServ as a policy methodology. In the case of Integrated services and RSVP, QPIM provides actions that control the reservation of the QoS requests within the network.

Four groups of actions are derived from action classes defined in [RFC3060] and [RFC3460]. The first QoS action group contains a single action, QoSPolicyRSVPSimpleAction. This action is used for both RSVP signal control and install actions. The second QoS action group determines whether a flow or class of flows should be admitted. This is done by specifying an appropriate traffic profile using the QoSPolicyTrfcProf class and its subclasses. This set of actions also includes QoS admission control actions, which use the QoSPolicyAdmissionAction class and its subclasses. There are three types of decisions a PDP (either remote or within a PEP) can make when it evaluates an RSVP request: a) admit or reject the request b) add or modify the request admission parameters c) modify the RSVP signalling content. The COPS for RSVP [RFC2749] specification uses different Decision object types to model each of these decisions. QPIM follows the COPS for RSVP specification and models each decision using a different action class.

The third group of actions control bandwidth allocation and congestion control differentiations, which together specify the per-hop behaviour forwarding treatment. This group of actions includes the QoSPolicyPHBAction class and its subclasses. The QoS actions modelled in QPIM can be used to control all of the building blocks of the Differentiated Service architecture, including per-hop behaviours, edge classification, and policing and shaping, without a need to specify the datapath mechanisms used by PEP implementations. The approach taken here is that a PHB action specifies both observable forwarding behaviour (e.g., loss, delay, jitter) as well as specifying the buffer and bandwidth resources that need to be allocated to each of the behaviour aggregates in order to achieve this behaviour. That is, a rule with a set of PHB actions can specify that an EF packet must not be delayed more than 20 msec in each hop. The fourth QoS action is an unconditional packet discard action, which uses the QoSPolicyDiscardAction class. This action is used either by itself or as a building block of the QoSPolicyPoliceAction.

Finally, the QoS policy information model specifies a set of pre-defined variable classes to support a set of fundamental QoS terms that are commonly used to form conditions and actions and are missing from the [PCIMe]. Examples of these include RSVP related variables. All variable classes defined extend the QoSPolicyRSVPVariable class, which itself extends the PolicyImplictVariable class, defined in [PCIMe]. Subclasses specify the data type and semantics of the policy variables.

The example shown below provides a set of rules that specify PHBs enforced within a Differentiated Service domain. The set of rules takes the form:

```
If (EF) then do EF actions
If (AF1) then do AF1 actions
    If (AF11) then do AF11 actions
    If (AF12) then do AF12 actions
    If (AF13) then do AF13 actions
If (default) then do Default actions
```

EF, AF1, AF11, AF12 and AF13 are conditions that filter traffic according to DSCP values. The AF1 condition matches the entire AF1 PHB group including the AF11, AF12 and AF13 DSCP values. The default rule specifies the Best Effort rules. The nesting of the AF1x rules within the AF1 rule specifies that there are further refinements on how AF1x traffic should be treated relative to the entire AF1 PHB group.

The actions of the corresponding PHBs can be any of the groups of actions described above. For example the AF1 PHB group policy action can be a Bandwidth action that defines the bandwidth allocation for the whole AF1 PHB group:

```
QoSPolicyBandwidthAction AF1:
     qpBandwidthUnits: %
     qpMinBandwidth: 30%
```

This work, presented by IETF on QoS policies, focuses on defining static policies for configuring DiffServ and IntServ nodes. Network-wide resource management policies as well as policies related to SLS Management are not addressed by this work, ignoring also completely the aspect of the constrained programmability achieved by using policies.

[Lymb03] presents a framework for specifying policies for the management of network services and focuses on solutions for dynamic adaptation of policies in response to changes within the managed environment. The term "*Policy Adaptation*" is used to describe the ability of the policy based management system to modify network behaviour in one of the following ways: a) adaptation by dynamically changing the parameters of a QoS policy to specify new attribute values for the run-time configuration of managed objects b) Adaptation by selecting and enabling/disabling a policy from a set of predefined QoS policies at run-time. The parameters of the selected network QoS policy are set at run-time. A usage scenario is also presented, where network policy that provides Per Domain Behaviour in a Differentiated Services environment is adapted by service management policies. Service management policies are enforced by Policy Management Agents at the service-level. The latter are responsible for the management of services that run within the managed DiffServ network. In the proposed framework, PDB policies are specified as Ponder obligation rules. The actual implementation of the PDB policy, i.e., the implementation of the PHB (or the set of PHBs) that will guarantee the QoS characteristics to the corresponding traffic aggregate, is hidden from the customer. The customer (human or automated

agent) is offered the externally observable PDB QoS attributes. An example of a PDB Ponder policy rule is given here.

```
inst oblig   /Policies/PDBPolicy1
subject      /PMAs/DiffServAgent;
target      r=/DiffServDomainA/Routers/CoreRouters;
on PDB1_ConfigRequest(DS, max input rate, min output rate);
do                      /* DS: The Diffserv codepoint for EF:
           101110.PDB1 is implemented with the EF PHB*/
r.applyEFPHB(DS, max input rate, min output rate);
when max input rate <D min output rate; /* Property that EF
                                    traffic must satisfy */
```

The work presented above, although it presents policies that can dynamically change the configuration of the nodes, it is restricted to policies for configuring DiffServ nodes in a scalable manner i.e. applying the same configuration in many network nodes. It failed to address important aspects of QoS Management such as Traffic Engineering and SLS admission control.

[Ston01] introduced a new language called the Path-Based Policy Language (PPL). PPL is designed to support policies that can be applied to both the differentiated as well as integrated services models proposed by the IETF. PPL provides a path-based representation of policies flexible enough to support both path- and non-path-based traffic flows. For example, providing an absolute path consisting of the links the traffic must take, will provide greater control over traffic flows and provide easier support to integrated services. A less specific policy may only need to provide source and destination nodes in its configuration, or perhaps just the specification that all traffic of, say, file transfers must be forwarded through a specific node acting as a firewall in an edge router. A summary of the constructs of the PPL language is shown below.

```
policyID<userID>@{paths} {target} {conditions} [{action_items}]
      policyID – unique policy identification token
      userID – user ID of policy creator
      paths – network paths the policy affects
      target – target class of network traffic
      conditions – any global conditions (items are AND'ed)
      action_items – for setting parameters (e.g., policy
      priority), declaring compromises and explicit deny, etc.
      action_item = [{condition}:] {actions}
```

The semantics of the PPL syntax are the following: a policyID created by <userID> dictates that target class of traffic may use paths only if {conditions} is true after action_items are performed.

Several examples of possible policies are provided in [Ston01] showing a wide range of policies that can be represented in this path-based approach. Using the wild card character of "*," the ability exists to represent explicit paths as well as groups of traffic flows with their language. This flexibility allows policies to be represented based on QoS and at the same time support existing best effort traffic. We provide below two examples of a policy specified in PPL:

```
Policy 1 <net_manager>@{<1,2,5>} {class = {faculty}} {*} {priority := 1}
```

This is a rule, which states that the path starting at node 1, traversing to node 2, and ending at node 5 will provide high priority for faculty users.

```
Policy2 <stone> @ {<*,2,*>, <*,4,*>} {*} time >= 1600, time <= 0800}
```

This rule states that all traffic will be allowed to traverse through nodes 2 and 4 during nonworking hours. Unless granted by another policy, traffic will not be able to traverse through nodes 2 and 4 during working hours. This is as a result of the default action, which is an explicit deny. Again this work is limited to static routing policies that are not dynamically changed (only based on time) and does not consider policies related to resource management and SLS management.

[Brun01] addressed the management of MPLS networks, following the IETF Policy Framework approach and extended the Common Information Model (CIM) for policies with MPLS specific classes. It proposed a three-level policy architecture including managing on device, network, and service level using policies, focusing on the network level policies for managing MPLS. On the network level, policies are concerned with Label Switched Paths (LSPs), including life-cycle management, LSP roles, LSP routing, and the mapping of traffic to LSPs.

The model of a basic LSP type contains an LSP identification, a reference to a Forward Equivalence Class (FEC), a reference to a traffic profile, a reference to a route specification, a role, and a resource class. The FEC specifies what type of traffic is using the LSP and the traffic profile specifies a resource profile for the LSP, e.g., bandwidth etc. The route specification may be used for explicitly set the route of an LSP or it is used to store the route chosen by the network independently of the policy server. The role is used to assign a certain property to an LSP from a management point of view. The resource class parameter specifies the class from which the LSP may draw the resources. Life-cycle management comprises setting up, releasing and updating an LSP along a number of routers. Additionally, the signalling processes may be controlled via policies using policy actions. The typical example comprises signalling of an LSP with QoS requirements, e.g. the signalled request for a bandwidth X may be permitted or refused, depending on the policies of the network domain. The mapping of traffic to LSPs is most likely performed at the edge router of an MPLS domain. However, in cases where tunnels for the purpose of traffic engineering are setup, the mapping may happen at any point in the network. Finally, for traffic engineering, LSPs can be used as tunnels for the aggregation of other LSPs, in order to explicitly specify the route the aggregated LSPs should take. In addition to LSP attributes, such as the resource class, a role can be assigned to an LSP. This helps classifying the LSPs in a domain such that for different roles different policies may apply.

An example policy rule, written in their proprietary policy definition language, looks as follows:

```
IF
LSR.TYPE contains WEIGHTED AND
LSR.NR_QUEUES gte 2 AND
LSR.Admin_state eq SETTING_UP
THEN
CONFIGURE SCHEDULER_WEIGHT {25%, 75%} AND
CONFIGURE QUEUE_LENGTH {20, 100}
```

The rule condition specifies that only interfaces with a packet scheduler able to perform weighted sharing of the link resource of 2 or more buffers is configured with the following configuration. The condition about the administration state tells the system, that only new LSRs are configured. The action specifies that 2 queues with length 20, 100 packets and scheduler shares of 25% and 75% are configured. The work presented above follows the IETF work on policies in the context of MPLS networks and suffers from the same limitations as described previously.

The framework defined in [Mart02] combines the IETF's Script MIB [RFC2592] and IETF's PDP/PEP architectures into a single architecture. Script MIB provides capabilities to transfer management scripts to distributed agents and to initiate and terminate the execution of the scripts. Since Script MIB can accept any form of management scripts, it provides support for arbitrary programming languages and multiple execution environments. Script MIB is used to communicate policies to Script MIB agents, which implement PDP functionality. The authors propose two solutions for deployment of policies within the proposed architecture. The first solution defines policies as programs. These are downloaded as scripts to the Script MIB agents and then executed by a common Script MIB runtime engine inside the agent. This runtime engine acts as a PDP and sends the corresponding low-level policy configuration to a local or remote PEP. A prototype implementation is provided for management of DiffServ Linux routers. Policies are implemented as Java programs; their actions involve creating and configuring Java objects that represent the DiffServ mechanisms of the routers according to the DiffServ MIB [Baker2001] data model. The second solution represents policies according to IETF's PCIM information model. Policies in this case are not defined as programs, but as groups of PCIM objects. These policy objects are downloaded directly to the managed nodes. Within each managed node, a policy interpreter implements PDP functionality to translate the PCIM policy objects to their corresponding low-level configuration commands. This work focuses mostly on implementation aspects of policy-based management using the Script-MIB and does not attempt to address the area of QoS Management policies.

[Ponn02] presented the design and implementation of a policy-based QoS management system for the IntServ/DiffServ based Internet which is based on COPS for interfacing with the network

devices and on LDAP for interfacing with a Directory for storing policies. The feasibility and performance of managing and deploying IntServ and DiffServ policies is demonstrated using Linux-based routers. They also presented some results from performance measurements related to the response time of the policy server for both COPS for RSVP and COPS-PR messages. This work is again based on the IETF policy framework and focuses on the implementation aspects and the performance evaluation of the protocols specified by the IETF for policy distribution. Another related work to policy-based management of DiffServ networks is described in [Fern01]. They propose a framework that offers QoS in a DiffServ domain using policy-based management and fuzzy logic techniques. This work does not focus on the policy aspects of the framework but mostly on the implementation of a fuzzy controller presenting results of end-to-end delay and jitter of the EF class for different network topologies.

Finally, a number of vendors are also marketing policy toolkits with most of them based on the IETF ideas. Cisco QoS Policy Manager [QPM] allows an administrator to define QoS policies that consist of a set of conditions and actions. The conditions represent a set of traffic characteristics that match a traffic flow while the actions are related to classification, limiting, shaping and queuing of traffic. Policies are defined through a web interface, stored in a database and are translated to device-specific Command Line Interface commands. In a similar manner, the Allot communications Netpolicy [Netpolicy] allows to specify policies that are stored in a directory for configuring devices using COPS, CLI or SNMP. Policies comprise conditions and actions where conditions are used for matching IP addresses, protocols, application data, type of service (ToS) settings and time of the day. The HP PolicyXpert [PolicyXpert] defines policy as a combination of one or more sets of rules. Policy rules consist of a single action and one or more condition lists. These are constructed from one or more conditions, which match against time/date or packet/traffic characteristics. Policy actions are used to manage DiffServ and RSVP mechanisms.

Conclusively, all of the above related work on QoS policies has focused on how to configure IP DiffServ networks through different policy representations but failed to address higher-level policies that guide the behaviour of QoS Management systems based on the business objectives of the operator addressing both Service Management and Traffic Engineering aspects.

2.6 Summary

In this chapter, we introduced the necessary background information required for the understanding of the rest of the thesis. We also presented related work in the context of QoS management and QoS policies since the work presented in this thesis proposes a policy-based approach for QoS management of IP networks.

More specifically, we first introduced general network management concepts by presenting the five network management areas and the manager-agent model defined by OSI-SM that are widely adopted standards by both the telecommunications and Internet management frameworks.

We then presented fundamental works on policy management frameworks and policy representations including the Ponder and the IETF policy frameworks. We then continued with some background information on the QoS models defined by IETF and gave a short overview of MPLS and Traffic Engineering.

Finally, we presented the most important related work on QoS Management architectures and QoS policies found in the literature.

Chapter 3

3 Building Extensible Management Systems using Policies

Chapter 3 of this thesis introduces our views and considerations on building extensible management systems using policies. It then discusses the salient characteristics of hierarchical management systems and proposes a framework for the coexistence of such systems with management policies. Finally, it presents a general framework for creating management services using policies based on pre-existing functionality in the system. The proposed frameworks are validated by relevant analysis in this chapter and are also backed up by case studies in the context of QoS Management of IP Networks presented in Chapter 5.

Policy-based management has been the subject of extensive research over the last decade focusing on the high level nature of policies used to guide the behaviour of a distributed system or network. Its flexibility also to dynamically change the configuration of the managed system has also been widely discussed in the literature. An introduction to traditional management systems of enterprise/Internet backbones and telecommunication networks is important since policy-based management can be applied to both management frameworks. While this presentation is based on relevant standards, it tries to shed light on some issues regarding inconsistencies that may arise in traditional rigid management frameworks, comparing and contrasting them with the approach of policy-driven systems where conflicts are the norm rather than an exception.

Policies, apart from their high-level declarative nature, can be also seen as a vehicle for "late binding" functionality to management systems, allowing for their graceful evolution as requirements change. This approach of using policies to build extensible management systems differentiates it from other presentations in the literature and as such, it constitutes a research contribution in its own right. In this chapter, we are exploring the potentiality of designing "policy-aware" management systems, in which a line has to be carefully drawn between "hard-wired" functionality and policy logic. Their relationship with other approaches used to achieve programmability in systems is also discussed and we show how we can achieve programmability in a constrained manner through policies.

The last two parts of the chapter propose two different generic frameworks, exploiting the strengths of the policy-based management paradigm. The first one explores the harmonic coexistence of management policies with traditional hierarchical distributed management systems, introducing different methodologies for their application to such systems. The proposed framework is essential for the understanding of the remaining of the thesis since it is applied to the policy-based QoS management architecture presented in Chapter 4. The last part of the chapter introduces a generic framework for creating management services using policies based on pre-existing entities/components. It is the policy-related interpreted logic that is dynamically introduced in the system that can be used to orchestrate and combine existing hard-wired functionality and create management services. The latter framework is again applied to components of the policy-based QoS Management architecture presented in the next chapter.

The structure of this chapter is as follows.

Section 3.1 presents an introduction to traditional management frameworks. First, the Internet Management framework, also called the SNMP management framework, is introduced and subsequently the Telecommunication Management Network is presented as the management framework for telecommunications networks. The presentation of the above frameworks identifies key issues regarding inconsistencies that may arise in both frameworks against which policy based management systems are compared in the next section.

Section 3.2 presents our views on policies starting with a discussion on the inconsistencies that appear in policy-based systems and we discuss the relationship of policies to mobile code paradigms and programmable network approaches. We then explore further the issue regarding which functionality of the system should be realised through polices and we try to provide generic guidelines for the design of policy-driven systems.

Section 3.3 presents a methodology for applying policies to hierarchical management systems and discusses the issue of policy refinement resulting into a policy hierarchy mirroring the system's management hierarchy. Section 3.4 introduces the concept of creating management services through a series of policy rules, combining functional components present in the system. The latter concept is inspired by similar approaches first introduced for service creation in Intelligent Networks (IN) platforms and service composition methods in programmable networks.

Finally, section 3.5 summarises what has been presented in this chapter, highlighting the research contributions.

3.1 Traditional Management Frameworks

One of the key motivations behind policy-based management is flexibility and graceful evolution of the management system so that it can adapt to changing requirements over a long period of time. This is achieved by disabling / modifying old policies and by introducing new ones in order to meet changing requirements. A key aspect of a policy-based management system is that changes to *targets* should be performed in a consistent fashion, avoiding policy conflicts that may leave the managed system in an inconsistent state. Conflicting actions do not occur only in policy-based management systems but are potentially possible in any control system, which performs intrusive management actions by *modifying* targets rather than simply *observing* them. Below we consider aspects of policies, intrusive management and conflicts in different traditional management frameworks.

3.1.1 Internet Management

This section discusses the characteristics of the Internet Management framework standardised by the Internet Engineering Task Force (IETF). This is also referred to as the Simple Network Management Protocol (SNMP) [SNMP] or the TCP/IP Management approach. Internet management has adopted the manager-agent model presented in chapter 2 and was designed to address the management needs of private data networks (LANs/MANs) and Internet backbones. As such, it has adopted a connectionless, polling-based, "remote debugging" approach. IETF following a pragmatic and result driven approach did not define any standards regarding the Internet Management architecture; only the management protocol (SNMP) and relevant Management Information Bases (MIBs) have been standardised. The design decisions behind the specification of SNMP were the following:

- Interoperability: all systems connected to the network should be manageable with SNMP
- Simplicity: the cost of adding network management to existing systems should be minimal
- Lightweight approach and small footprint on agents
- Robustness

Its simplicity has been the main reason for its wide acceptance and deployment in almost every managed network device.

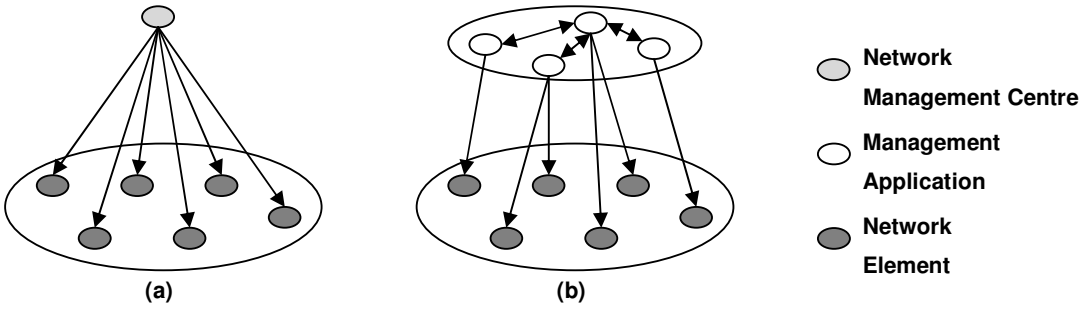


Figure 3-1 (a) Centralised and (b) Flat Management Models of Management Organisation

The SNMP architecture is a relatively simple management architecture consisting of a single, centralised "network management centre (NMC)" – an application acting in a manager role. The latter supervises network elements (agents) located typically in a cluster of local / metropolitan area networks. SNMPv1 exemplifies best the centralised model of management organisation (Figure 3-1a) while SNMPv2 introduced manager to manager communication capabilities reflecting the flat model of management organization (Figure 3-1b) with management applications acting in a dual manager-agent role. In the Internet management architecture, the elements are typically configured one-by-one, in an isolated fashion, through the supervision of a human network manager and according to an overall network operation policy, which is worked out beforehand. This means that (re-)configuration is infrequent and takes place manually. In an evolution of this scheme, configuration parameters for every device are stored in a repository e.g. a directory, which is contacted by the devices upon cold or warm starts so that a device picks up necessary parameters and configures itself; this makes the system more scalable.

The NMC supervises, i.e. monitors, the managed devices, provides a view of the current network state, alerts the human manager in case of abnormal changes but does not attempt to reconfigure the network using automated logic. Reconfiguration is typically left to human managers who may modify first the network operation policy using their intelligence to overcome the problem. This approach is comparable with using debuggers to debug or "manage" computer programs. Ordinary debuggers allow programmers to watch and modify program variables. A debug program however does not automatically determine which variables should be monitored and which modifications must be made. Such decisions are made by the human programmer; the debugger provides the means to access the program variables. In the same way, the Internet management architecture follows a "remote debugging" approach, providing the ability to managers to remotely access and modify management variables. The decision on which variables to be monitored and which modifications to be made is left to the human administrator. This decision reflects the operator's business objectives and network operation policies.

There are no conflicts in this architecture, inconsistencies might only occur because of a wrong human-derived configuration policy but, with the right precautions, this should not happen. One of the main disadvantages of the SNMP framework is that an enormous amount of management variables has been defined while lacking at the same time a good functional structure to classify them. To determine which variables to be watched and what modifications to be made requires that human operators should have a good understanding of the precise meaning of the managed variables. It is very unlikely that many people will have ready sufficient knowledge. As a consequence, network management may therefore become time consuming and prone to inconsistencies due to erroneous configurations. While quite simple as presented, centralised architecture with emphasis in monitoring rather than control, works adequately for best-effort IP networks, it cannot meet the needs of emerging multiservice networks with QoS guarantees. The latter require frequent, automated configuration changes according to a network-wide view, as it will be presented in Chapter 4.

3.1.2 Telecommunications Management Network

Telecommunication networks are managed according to the hierarchically distributed Telecommunications Management Network (TMN) model defined by ITU-T [M3010]. OSI-SM has been adopted as the base technology for TMN, incorporating most of its basic ideas i.e. the manager-agent approach, the object-oriented paradigm, taking also into account the five OSI functional areas (FCAPS, i.e. Fault, Configuration, Accounting, Performance and Security management). Within the general TMN architecture, there are three aspects which have been considered when designing and planning a TMN:

- The *Functional architecture* which defines the appropriate distribution of functionality within the TMN in order to allow for the creation of functional blocks and reference points between them
- The *Information architecture* which gives the rationale for applying the OSI-SM principles to TMN extended to fit the TMN environment where appropriate.
- The *Physical architecture* which describes realisable interfaces and interoperable physical components that make up the TMN.

In the rest of the section, we will examine the logical layering of the TMN architecture since it is this particular aspect of the TMN based on which a lot of work in this thesis is influenced. More detailed descriptions of TMN can be found in [Pavl94][Pavl96][Pavl98].

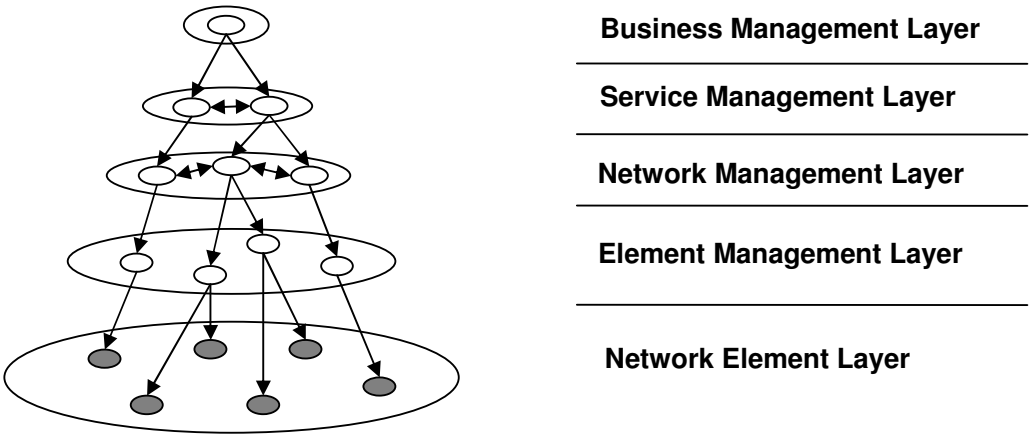


Figure 3-2 Hierarchical Distributed Management Model and TMN Management Layers

One of the key aspects of the TMN as a management framework is that it transforms the old *centralised* or *flat* management paradigm adopted by the Internet Management architecture to a hierarchical distributed model of management operation (Figure 3-2). In the hierarchical model projected by the TMN, management functionality is layered, offering increased abstraction and encapsulation in higher layers. The functionality of each layer builds on the functionality offered by the layer below. The manager and agent roles are not fixed and management applications may act in both roles in either peer-to-peer or hierarchical relationships. Management functionality may be organised in different layers of management responsibility: element, network, service and business management according to the TMN model.

The *Element Management Layer (EML)* manages each network element on an individual basis and has a set of element managers that are individually responsible, on a devolved basis from the network management layer, for some subset of the network elements located in a small geographical area. EML comprises functions concerned with day-to-day management of network elements in terms of configuration, faults and maintenance, performance monitoring etc. An abstract view of the elements is presented to the network management layer so that the latter is shielded from unnecessary complexity and detail. Finally, EML provides also a gateway (mediation) function to permit the network management layer to interact with the network elements.

The *Network Management Layer (NML)* comprises functions that address the whole network spanning a large geographical area e.g. a country. It is concerned with the coordination and control of all the network elements within the scope of its domain, keeping the state of the network at some operational optimum. NML usually contains a logically centralised but physically distributed network manager. The latter has a view of the network-wide policy and implements it through automated logic by supervising the network elements and reconfiguring them in order to introduce new services or to recover from performance, fault and other problems.

NML interacts with the service management layer on performance, usability, availability etc providing the "networking" demands of the services offered to customers.

The *Service Management Layer (SML)* is concerned with the contractual aspects of services provided to customers. Its functions comprise customer facing and interaction with other administrations in order to support end-to-end services. Additional tasks include interactions with the NML and the business management layer, and the control of interaction between services, service ordering, complaint handling and invoicing. At this layer a technology independent view of the network is possible which usually is treated as a "cloud" with certain capabilities.

The *Business Management Layer (BML)* is responsible for carrying goal setting tasks and is concerned with the implementation of policies and strategies within the organisation which owns and operates the services and possibly the network itself. Business decisions may be influenced by higher-level controls such as legislation or macro-economic factors. They may address charging policies, guidance on establishing customer or peer provider agreements, service and network operation strategies and so on. Although not considered at the time the TMN management layers were conceived, policy-based management attempts to provide the means to automate some of the functionality of this layer since it facilitates the setting of high-level goals and policies of the operator in a formal human-friendly way. The goal achievement functionality is not part of this layer as it is later taken care by the policy system that translates the defined goals and performs the appropriate management operations on components of the layers below in order to realise the operator's policies. It is the work presented in this thesis that provides a complete methodology to apply policy-based management in hierarchical management systems as the TMN-influenced QoS management system proposed in the Chapter 5.

Management logic of the management applications present in all the layers of the TMN architecture can be altered to a limited extent by modifying managed objects that model their operation. An example of TMN-influenced proactive and reactive management systems for ATM management can be found in [Georg99]. The latter is concerned with the initial configuration and ongoing dynamic management of the ATM virtual path (VP) layer. Specifically, it hosts dynamic virtual path connection (VPC) bandwidth management, VP layer design and dynamic reconfiguration, fault management (filtering and correlation) functions, as well as generic configuration and network resource monitoring functions. All configuration changes occur through a configuration manager, which holds the physical and logical network topology and partitioning. Requests coming from service, performance, fault and other managers are carefully validated, in order to maintain network consistence and integrity. Despite this validation, it is possible that different managers have conflicting configuration requirements. This can lead to inconsistent network state that satisfies only one of them, or to race conditions, in which the managers keep requesting changes to their preferred configuration state when they sense it has

been changed back. Such conflicts can be avoided by careful modelling, designing and testing the management system, but conflicts may still occur at run-time when the system is stressed by real-world conditions not previously anticipated. This is rare though and also points to system integrity issues which are outside the scope of this work.

3.2 Policies as a Means for Extensible, Programmable Management Systems

Policy-based management may be applied to both enterprise/Internet and telecommunication networks. The view taken by IETF seems to be compatible with the centralised model used in managing enterprise networks, though policy work in the research community has previously pointed to distributed models. In this discussion, we will consider the centralised model to demonstrate the points and we will examine policies in distributed hierarchical systems in the next section.

3.2.1 Inconsistencies in Policy-driven systems

The key aspect of a policy-based system is that management logic is expressed through declarative policies, evaluated in policy consumers. In the IETF model, the policy consumer can be thought as a centralised manager, with the execution of provisioning policies resulting in configuration operations on managed objects within network elements. In [Slom99], the policy consumer is seen as a hybrid manager-agent where the policies express the manager intelligence and access the co-located managed objects of the agent part; we consider and extend this model in the next section. The essence though is that management intelligence can be modified, added and removed by manipulating policies as requirements change. In policy-based systems, management intelligence does not follow the rigid analysis, design, implementation, testing and deployment cycle, and as such, conflicts may be the norm rather than the exception. Conflict analysis and detection is required both statically, at policy introduction and deployment time, and also dynamically, at run time. Policies are often associated with interpreted logic but we believe their salient characteristic to be the composition of a system from building blocks which can be introduced, modified and withdrawn at any time, without having rigorously tested the resulting system in every such modification.

Taking the policy approach to the extreme, all management intelligence could be policy-based, starting with a system which comprises only manageable network elements and policy consumer capabilities in the role of an “empty” centralised manager. This is the complete opposite of the rigid, hierarchical TMN approach, but results in a pretty undefined and fluid system, which will be very difficult to protect against conflicts, or to even realise it with declarative policies in the

first place. We see policies mostly as a means to "late bind" functionality to an existing management system, which is hierarchically distributed in order to meet the management needs of multiservice networks. In this case, policies can be seen as a means to achieve "programmability" of the system with "new" functionality and lead to a flexible system that can cope with evolving requirements. We feel this is a much more realistic proposition than a purely policy-based approach for complex management systems.

We discuss below the programmability aspects introduced by the policy-based management paradigm and we compare it with existing approaches in the literature for active/programmable networks and mobile code paradigms.

3.2.2 Relationship to Programmable Systems and Networks

The main drive behind programmable networks and systems is the idea of developing communication networks and systems that exhibit high degree of flexibility, extensibility and customisability. There are many mechanisms supported for programming network components categorised into two main categories [Tenn97]. The first approach known as *capsules* or *smart packets* considers packets as miniature programs that are encapsulated in transmission frames and executed at each node along their path. The second approach is called the *programmable switch* or the *active node* which provides a mechanism to download programs into network nodes. Mobile code paradigms were also adopted in the area of Network and Systems Management in order to provide flexibility to management systems by dynamically transferring programs into agents and having them executed by them. [Fugg98] identified three types of mobile code paradigms. The first one is called Code On Demand (COD) where a client downloads (pulls) required code and initialises it in order to perform a task. The code is downloaded from a "code server" where software components are stored. The second one is the Remote Evaluation (REV) where the client uploads (pushes) the code containing the required logic along with initial parameters to a remote server for execution. Finally, the third one is the Mobile Agent (MA) paradigm where an execution unit containing the logic to perform a task is able to autonomously migrate to a remote node and resume execution seamlessly.

Most of the work on Policy Management, especially from the IETF community, emphasises on the high-level aspect of policies since the basic problem they were trying to tackle was the low-level granularity of SNMP and the heterogeneity of the network devices. IETF adopted the Policy-based Management paradigm as it was seen as the panacea to the problem of configuring network nodes, providing the ability to the network administrator to configure the network in a high-level and device-independent manner. Although the latter is one of the salient characteristics of policies, their programmability aspect was never discussed extensively in the research

community, which differentiates it from a simple high-level configuration of managed elements. [Slom99] discussed first this relationship but focused on how to use policies to complement the approaches to programmable/active networks described in the previous paragraph. Specifically, obligation policies were used to define the events, constraints on loading or executing a code on a programmable agent and authorisation policies were specified to define who can program specific components and what programming operations they can access.

The widely accepted definition of policies was first introduced in [Slom94] as rules governing the behaviour of a system. Policy-driven systems should be able to adapt to changing requirements and dynamically change their behaviour based on newly enforced policies without having to recode components of the system. The general format of a policy rule, both adopted by the IETF and the research community as presented in the Chapter 2, *on event if* <*conditions*> *then* <*actions*> represents some sort of management logic rather than a simple configuration operation. The broad definition of distributed systems management is to monitor the activity of the system, make management decisions and perform control actions to modify the behaviour of the system. Policies allow these management decisions to be performed when certain conditions hold, realised by logic dynamically introduced in the system. In comparison to traditional protocol-based management frameworks where they provide only the means to monitor specific attributes e.g. through a GET operation in SNMP and to modify their values e.g. through a SET operation in SNMP, this logic is either hardwired in management applications or the human operator takes the decisions and realises them by performing the corresponding actions after he/she has worked them out in his mind.

Consequently, in order to design and implement a policy-driven system, code has to be generated and executed dynamically, performing operations either locally or remotely on the managed objects of the target entity. This code should implement the policy introduced by the administrator, reflecting his management decision in a low-level, policy target-specific format. Note that policies do not result in enhancing the target system functionality i.e. the functionality supported by agents in managed elements but they represent *management* logic that conceptually belongs in the manager application. Enhancement of the agent functionality e.g. router functionality can only be achieved by approaches used in active/programmable networks. Moreover, in policy-based management systems, code does not have to be transferred and executed in remote entities but it can be executed locally in the components responsible for translating the high level notation to code i.e. the policy consumers or PDP in the Policy Framework. The latter is an important advantage of policy-based systems in comparison with the mobile code approaches regarding security risks.

Management by Delegation (MbD) was the first management architecture based on the mobile code management paradigm introduced by [Yemi91]. The MbD approach is based on delegation-

agents, programs that can be linked and executed under local or remote control. Delegation-agent programs can be written in arbitrary languages, interpreted or even compiled. The Script MIB [RFC2592] standardised by the IETF DISMAN working group follows the MbD approach, allowing managers to distribute tasks to agents in the form of scripts using SNMPv3. Similar approaches have been adopted in order to implement policy-based systems, with policies downloaded to policy consumers where they are eventually translated to interpreted scripts which are dynamically executed, performing the required management operations [Mart02] [Marr96] [Flegk03]. Policy-based management frameworks can support as described in chapter 2 both the outsourcing model (pull) and the provisioning model (push) as defined by the IETF in a similar manner as the REV and COD approach in the mobile code systems. Finally, the mobile agent paradigm has also been used in policy-driven systems, either as a propagation mechanism for policy rules [Kim03] or as the target system whose behaviour is controlled via policies [Corr01], [Yang02].

From the above it is obvious that there exist many approaches to achieve programmability in a system, with policies being one of them. Although the main goal of all the approaches is to build flexible and extensible systems, they differ on the level of programmability they provide. The level of programmability expresses the granularity in which new functionality can be introduced in the system. One can consider a spectrum of possible choices from highly dynamic to more conservative levels of programmability. At one end of this spectrum, mobile code paradigms enable the uncoordinated deployment of new code/functionality in the system and provide the most dynamic approach. At the other end of the spectrum, the traditional manager-agent (client-server) paradigms in which changes in the behaviour of the system are only allowed by performing operations on managed objects constitutes the most rigid and conservative approach. The level of programmability has a direct bearing on safety and security, speed, performance and flexibility of the system. Although mobile code offers the highest degree of flexibility, it has not yet been widely adopted. Security has been the major obstacle to its wide deployment, since execution of code with malicious and inadvertent bugs can harm and eventually destroy the system. Below, we argue that policies offer programmability in a more controlled manner, providing a more constrained means for adaptive behaviour of components.

Policies after being defined in a high-level declarative language and stored in the repository, they are eventually translated into code which is dynamically executed performing operations on the managed entities by policy management agents/policy consumers. The capabilities/logic to generate this code in the policy consumers is constrained to pre-conceived policy types which were conceived at the requirements engineering phase of the system design. The formal representation of policies, either in declarative languages found in the literature or in object oriented formats, is important in order to facilitate the realisation of such kind of logic. This logic

is in fact hardwired in policy consumers and follows the rigid analysis, design, implementation, testing and deployment cycle as the rest of static intelligence of the system. The latter implies that with careful design it is very unlikely that erroneous or malicious code will be produced that can harm the system. In contrast, mobile code paradigms allow the execution of arbitrary code usually retrieved from code servers where software components are stored. Of course, this constrained programmability offered by policies leads to a less flexible approach since the "new" logic introduced has actually been pre-conceived but is not yet instantiated in the system.

Policies in network management have been used extensively in the literature for driving the behaviour of the network. We view policies as a means of programming network management systems as well as network devices. Policies enforced on a management application operating in a Network Management Centre i.e. on a general purpose computer outside the network, are an example of the former category, giving us the means for a more flexible management systems that can be driven according to operational and business requirements. Programmability through policies is important in this case since it effectively results in a dynamic change of the management system's behaviour but in a constrained manner as described above. One might argue that this new functionality could also be provided by actually re-programming the management application. This "hard" programmability is more costly in terms of the overall required cycle until deployment but gives ultimate flexibility. Policies that are downloaded and executed in the routers, allowing to drive the router's behaviour constitute an example of the latter category. In this case, programmability is very important since it results in dynamic extension/modification/withdrawal of the router functionality, something hard to be done otherwise, with routers being typically "closed boxes". This programmability is again constrained to the types of policies that have been conceived at the requirements engineering phase. In this case policy consumers reside *inside* the routers producing and executing locally the code realising the policies. The latter approach reduces significantly the security risks compared with the approach where a manager remotely sends the code to be executed in the nodes, for which they provide an execution environment through a programmable interface.

Conclusively policies, although constrained to pre-conceived types, provide a controllable framework for functionality extension that can be harnessed further through policy conflict detection. In the section below, we propose generic guidelines for designing policy-driven systems, highlighting the situations where logic of the system should be realised as policies.

3.2.3 Guidelines for the Design of Policy-driven Systems

Policy-based management provides the ability to dynamically configure a system, by separating the rules that govern a system's behaviour from the functionality supported by it. We view

policies as complementing the static management system intelligence. One key aspect when designing a policy-capable management system is how much intelligence should be realised in a static fashion. Static intelligence should offer enough functionality to allow relatively easy extension of the system through policy logic but not too much so that there is still flexibility in terms of changing requirements. In principle, higher amount of static intelligence leads to a more rigid, less extensible but potentially more stable system while less amount of static intelligence leaves the system fluid, easily extensible but may result in instability as more and more functionality is realised through policies (more frequent conflicts etc.). Since policies are often associated with interpreted code, performance is also key issue to be considered besides the stability of the system. The main drawback of interpreted code is the cost of processing involved with code interpretation (which typically is significantly higher than the cost of binary code execution). Consequently, the more logic realised through policies, the more resource consuming the system becomes, with significantly degraded performance in terms of code execution time. Below we try to address this issue of where to draw the line between static system functionality and dynamically introduced logic through policies.

As described in the previous section, the logic present in the policy consumers for generating the code realising the policies is hardwired and based on policy types that were conceived at the requirements engineering phase of the system's design. One of the key aspects of requirements engineering is the elicitation of goals to be achieved by the system envisioned i.e. why the system is needed based on current and foreseen conditions [Lams01]. At the beginning of this phase, high level goals are defined which are more strategic, coarse-grained and are later refined into lower level concrete goals realised by existing or to-be components of the system. The next stage is to operationalise these goals in the software components of the system by operations supported by the functionality of these components. This refinement of the high-level requirements (goals) to lower level ones and the operationalisation of the latter are the important phases in the system's design, where the architect should take the decision of which functionality to be realised as policies. We propose the following guidelines to determine potential situations in which a policy-based implementation would be appropriate:

1. When the refinement of the high-level goals of the system results into a disjunction of sub-goals (i.e. the high-level goal can be achieved by one of an OR-decomposed set of sub-goals), the management logic that decides which of the sub-goals to satisfy i.e. the set of operations to be invoked realising the subgoal, should be realised as a policy. This means that system's (static) functionality should include all the operations for satisfying the subgoals identified, but the decision which one to choose is left to be made later by the user of the system. The latter i.e. the administrator in case of a management system has the flexibility to drive the behaviour of the system by encoding this decision into a

policy together with the conditions on when to apply the above solution. Although all the lower level goals satisfy the higher-level goal, each one might exhibit different behaviour.

2. When the operationalisation of a (low-level) goal results into multiple solutions (sets of operations) to be supported by the system in terms of functionality. These multiple solutions should follow an OR-decomposition for the same goal. Again this decision should be left to be encoded as a policy defined later by the user, reflecting his/hers business objectives. Again here, the resulting alternative solutions to realise a low level goal might represent different system behaviour.

3. When the operationalisation of a goal results in parameterised operations whose parameters the user of the system would be interested to change at a future point in time. The value of these parameters should reflect the current user's requirements at the time, and providing that these operations can be encoded into policies, the user can dynamically introduce the logic in the system on what values to set these parameters based on the current system state/condition.

The above constitute generic guidelines on what logic to be encoded as a policy in a system. Additionally there may be application-specific guidelines that further guide the user in their decision to apply policies.

As it can be observed from the above, policies do not result in enhancing the target system's functionality since they provide only the required (management) logic on when to trigger predefined choices of the behaviour of the system. These behavioural choices are static hardwired operations supported by the system functionality. One could think that the operations needed to satisfy a goal could be entirely generated by the code realising a policy but this does not add any extra flexibility in the system since the latter has been preconceived when designing the system. Consequently, there is no additional benefit realising the above functionality through policies and will probably lead into a less stable and resource demanding system.

Until now it is not clear how policies can be used in the context of a hierarchical system. In the next section we consider policies that follow and mirror the hierarchical system decomposition.

3.3 Policies in Hierarchical Distributed Management Systems

In hierarchical management systems, hybrid agent-manager applications exist at different levels of the hierarchy, managing ultimately network elements at the lowest level. Manager-agent or managing-managed interactions occur top-down and possibly peer-to-peer but never bottom-up. A hierarchy may be *strict*, in which case the management layer N+1 builds on the functionality and

services of layer N, or *relaxed*, in which case layers may be bypassed. In the following discussion we will assume a strict hierarchy for simplicity.

At layer N of the hierarchy, an agent-manager application comprises:

- Managed objects presenting the management capabilities of the application to the layer N+1 (or to the same layer N for peer-to-peer interactions).
- Management logic accessing managed objects of the layer N-1 (or of the same layer N for peer-to-peer interactions).

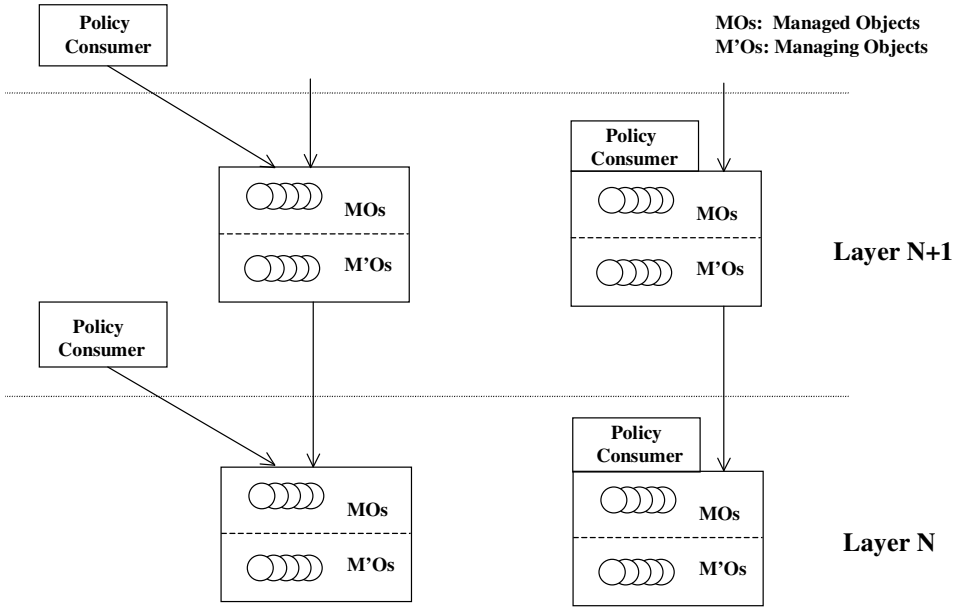


Figure 3-3 Hierarchical management with loosely (left) and tightly-coupled (right) policy consumers.

The managed objects constitute the top, i.e. agent, part of the application (see Figure 3-3). The managed objects and associated managing logic or managing objects represent "static" management intelligence, following a rigorous analysis, design, implementation, testing and deployment cycle. Parameterisation of the functionality of such an agent-manager application is possible to a limited extent by configuring managed object values. The deployment of such a hierarchy takes place bottom-up but the decomposition and design of the whole system takes place top-down, according to the management services to be provided.

The simplest form of introducing policies in such a system is through a separate policy consumer point where policies execute and access managed object at various different layers of the management hierarchy. In other words, the policies manipulate targets, i.e. managed objects, at all the layers of the hierarchy. The problem with this approach is that the policies are monolithic, logically and physically centralised, operating on a hierarchical distributed system. A better approach would be to structure the policies hierarchically, mirroring the system hierarchy, as explored next.

In a hierarchical policy system, policies at layer N+1 operate on managed objects of layer N. This implies in fact that these policies may be considered as part of the managing intelligence of layer N+1, in addition to the static intelligence of agent-manager applications. This approach is shown in the left of Figure 3-3. If these policies access managed objects in more than one layer N agent-managers, they could execute at a layer N+1 consumer point which complements the managing intelligence in this layer. If though they access managed objects in a single subordinate agent-manager, they could execute *at* that agent-manager, having local access to managed objects. In this case, the manager-agent at layer N is programmed with policy logic that belongs conceptually to the layer N+1 but since it relates to a particular agent-manager of the layer below, it has actually "migrated" there. This is shown in the right part of Figure 3-3 where the two policy consumers have migrated and now form an integral part of the agent-manager they relate to. In this paradigm, every agent-manager may potentially become a policy consumer, including of course ultimately the agents within network elements.

A key aspect in such a hierarchical system is policy refinement and this should naturally follow the hierarchical composition of the system. Policies may be introduced at any level but higher-level policies may possibly result in the introduction of related policies at lower levels. In a similar fashion to the bottom-up deployment of a static hierarchical system, policy hierarchies should be introduced in a bottom-up fashion, maintaining the completeness and integrity of the policy space. [Moff93] introduced the notion of policy hierarchies where high-level policies are refined into a number of more specific policies forming a policy hierarchy. Several types of relationships have been identified between the policies present in the hierarchy, depending on different types of derivation of the lower level policies. This derivation can be done by refining goals, partitioning the targets of the policies or delegating responsibilities to other policy managers/consumers. Below we explore further the concept of policy hierarchies in the context of hierarchical management systems.

In strict hierarchical management systems, management layer N+1 provide guidelines for the operation of management layer N. Policies enforced at layer N+1 drive the behaviour of the agent-manager applications of that layer and consequently the operations performed from the managing objects of layer N+1 to managed objects of layer N and so on. The effect of the layer N+1 policies is reflected in the guidelines produced by it and as such layer N and lower layer policies may not be needed if the operation of the layers is bound to the constraints produced by the layer above. Conclusively, a policy enforced at layer N+1 is reflected in all the layers below until the target managed elements by the manager agent operations occurred between layers and performed by the static functionality of the system. When though management layers operate in a less constrained fashion, possibly ignoring some of the guidelines of the higher layers, an introduction of a policy at layer N might result in the introduction of policies in the lower layers

of the management hierarchy. The latter is necessary since the behaviour of a layer does not always adhere to the guidelines of the layer above and might lead the system to a state which does not satisfy the introduced policy.

For example, in a strict hierarchical TMN-based system, agent-manager applications in the element management layer operate within the constraints set by the network management layer of the system. A policy enforced at the network management layer is sufficient to realise the desired behaviour in that layer and all the layers below since the guidelines produced for the element management layer and consequently the operations of the element management layer on the network elements reflect the network management layer policy. On the other hand in a less rigid TMN-based system, policies might be introduced both in the element management and network element layers since element managers might not act within the constraints set by the network management layer and consequently the configuration of the network elements will not reflect the introduced policy. Of course the latter approach leads to a more flexible system but with a higher risk of leading the system into a less stable state as argued below.

As you go up the layers of the management hierarchy, agent-managers have a more abstract view of the target managed elements at the lowest level of the hierarchy. It is common practice that in higher management layers, the level of distribution of the desired functionality in terms of agent-managers is decreasing, leading eventually to a single centralised management application at the highest level of the management hierarchy. Moreover, higher levels of management hierarchy operate in a longer time-scale usually based on static information while lower level management layers functions operate in a more dynamic fashion. Using a TMN-based system as an example, an application residing in the network management layer has a centralised network wide view while element managers have a narrow uncoordinated view of a small number of network elements for which they responsible to manage. Hence, if the element management layer utilises the guidelines produced by the network management layer, this will probably result in a more stable overall system. On the other hand, these guidelines are produced based on static information, which can be quite different from the actual state of the network, providing reasonable grounds to ignore them in special cases. The latter situation will add more flexibility to our system since every layer will be guided by distinct policies introduced according to the operator's requirements but, unless there is some coordination of policies in the layers, this may result in an unstable state for the whole system.

The above case introduces another level of policy refinement than the one presented in the literature i.e. the refinement of abstract high-level goals. The latter deals with the process of transforming a high level, abstract policy specification into a low-level concrete one that the system can enforce. In the case of hierarchical management systems the refinement of high level policies might not stop when they become operationalised i.e. can be supported by the system

functionality but should also be refined to policies enforced in every layer of the system hierarchy as described above. Policy refinement and transformation is a process analogous to software system analysis and design and is almost impossible to completely automate the refinement process without any human intervention. However, in the context of a hierarchical system of a specific nature, application specific patterns may be identified that will maximise the level of automation of the refinement process. We are in fact envisaging situations in which changing parameters of a high level policy will result in changes throughout the policy hierarchy.

The above proposed methodology for applying policies in a distributed hierarchical system will be validated by its application on a management system for IP DiffServ networks presented in Chapter 4, which was designed with policy extensibility in mind. In the next section, we explore further the powers of the policy-based management paradigm and we present a generic framework for management service creation through policies.

3.4 Management Service Creation through policies

Network management systems support management services, which are primarily used by the operators and human managers of the network. They may also support services that are used by the end-users such as customer subscription, accounting, service profile customisation etc. In traditional management systems, these services are fixed in the sense that new features can only be added after a full research-standardisation-deployment cycle and are static as far as their use: they execute according to their built-in logic and users (operators) may customise their operation only through tuning standardised operational parameters before service execution. Management services are usually created by assembling individual management service components. The latter are further decomposed into management functions which are mapped onto managed object classes through an object modelling phase.

Service creation techniques were first introduced in the context of Intelligent Networks (INs) [Q1200] for supporting telecommunication services. Rather than creating new services by writing new software code, modules of service independent logic can be arranged using graphical user interfaces in an abstract way. The user uses the graphical editor to add building blocks to the picture and to link them together. Once the picture is complete it is then passed to a module which generates code conforming to the application programming interface (API) supported by the IN platform. Finally, a set of platform and network specific libraries is linked to create the final service logic. The complete service logic program is then ready for offline testing and eventual deployment to an IN node. The advent of programmable/mobile code paradigms opened new possibilities, allowing dynamic, customisable and more flexible service creation to the users of the system. The latter approaches provide ultimate flexibility since they support extensibility of the

system's functionality at run time, introducing in this way new services in the network. In the active networks realm, a service creation/composition method provides a means to specify a composite service constructed from building blocks called components. As such the realisation of a composition method is a programming language whose primary concern is to operate on components. [Silv98][CANE][Hick98] are examples of composition methods that enable the user to develop services using a high level programming language and deploy them dynamically in nodes of an active network.

Until now, most of the work on policies focused on how to drive the behaviour of the managed system though high-level declarative directives introduced by the system administrator. In this section, we describe a framework/methodology on how to create a management service based on predefined entities/components using policies. As we have already explained in previous sections of this chapter, policies are eventually translated into interpreted scripts that realise management logic, which is dynamically introduced in the system and is executed at runtime. Although this logic can change the behaviour of the system, it is strictly dependent on predefined choices, realised by the functionality encapsulated in the Managed Objects of the managed entity as well as in the translation logic responsible for transforming the policies into interpreted scripts. Conditions and actions of the policy rules result in interactions with variables and methods offered on the boundaries of these managed objects comprising the required logic needed to combine this functionality present in the target entities.

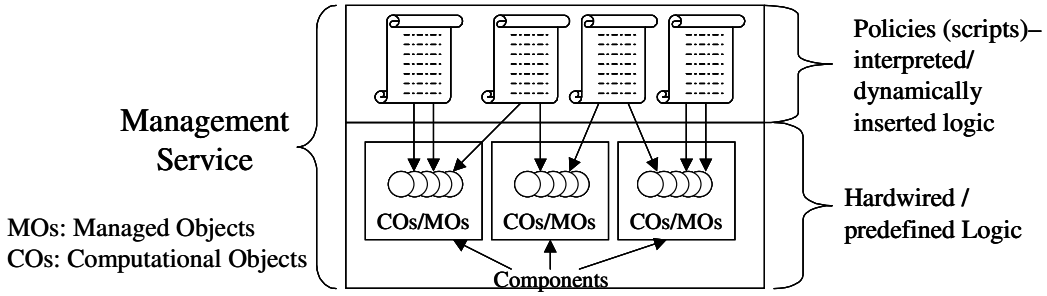


Figure 3-4 Management Service creation based on policies and predefined components

It is this combination of *both* the hardwired/predefined logic together with the dynamically introduced logic that can create a new management service. Since we followed a policy-based approach for realizing this management logic, it is the policies defined by the administrator that trigger and bind together the hardwired functionality of the system for creating a new management service. Moreover, the high-level nature of the policies gives to the network administrator the ability to "program" the new management service using a high-level declarative language instead of low-level scripting languages. This is also done in a scalable fashion since assuming a distributed managed system, policies can be delegated and executed in all the managed entities the policies apply to (tightly coupled approach shown in Figure 3-3). The latter

will result in simultaneous deployment of the service to all the target entities. Of course this means that these managed elements have all the necessary functional components/capabilities for the policies to be enforced. Policies are introduced through a policy management tool offering an equivalent service creation environment and are stored in the policy repository according to a predefined schema. They are then translated to interpreted scripts and executed on the fly. The last two components are not shown in Figure 3-4 but have been extensively described in Chapter 2.

As shown in Figure 3-4, the hardwired functionality of the system can be modelled using a component-based approach. Typically components can be defined as independent packages of elementary software services. It is worth noting that component-based system development can co-exist with object-oriented analysis, design and development, which is the standard methodology for developing management systems. While this methodology focuses on analysing and designing systems for a specific problem, at the same time it is useful to perform the packaging of the developed system into a set of self-descriptive, application independent components using the modelling constructs of a component technology. The components are furthermore decomposed into Computational Objects (COs), which define separate functional units and/or different methods for realising this functionality, providing in this way different choices/alternatives to achieve the same functionality. The choice of which behaviour supported by these objects would be triggered is left to the relevant policies. When developing such a system, the components present should be defined at the requirements capture and analysis stage while the actual management services would be created at a later stage when the administrator defines the relevant policies. As in all policy-based systems, this service creation logic does not follow an analysis, design/development, testing and deployment cycle, which renders policy analysis functionality essential i.e. conflict detection and resolution mechanisms. The definition of the line between static and policy-based intelligence still remains an open research issue.

Policies for creating management services result in interactions with the computational/managed objects of existing components. Enforcing a policy rule i.e. evaluating the conditions and executing the actions, usually results in operations on objects that belong to the same component; it is also possible that policies result in registering events and invoking methods that belong to different components, combining in this way the component functionality. Although, in Figure 3-4 it is shown that policies are delegated and evaluated at the managed entities following a completely distributed approach, they could also be executed by an application that is not attached to the managed entities and invokes remotely operations on managed objects in different managed elements.

One of the important features of service creation methods is the control of the order of execution of the components termed *sequence control*. When sequential execution is required for a service i.e. execute component A in its entirety followed by component B, then policies triggering

components should be enforced in the same order. This can be realised by specifying rules that their evaluation and execution are based on events and conditions that hold only after a component has finished its execution. Moreover, data that are produced as a result of execution of a component can be passed as input to other components enabling this way data sharing between components. On the other hand rules that do not have this type of relationship can be executed concurrently supporting the parallel construct of service composition.

Finally, management service creation through policies can benefit from policy refinement techniques that may assist the administrator to identify the set of policy rules needed to be enforced in order to create a service. Management services can be defined as abstract high-level goals which can be refined to lower level ones realised by a number of policies. Policy refinement can aid the administrator to choose between alternative behaviour of the components needed to be combined and automate as much as possible this derivation of the low level policies leading to a less error prone service composition.

3.5 Summary and conclusions

In this chapter, we first introduced the Internet Management Framework defined by IETF used to manage Enterprise Networks and Internet backbones and the Telecommunications Management Network (TMN) framework for managing telecommunication networks. The description of the above frameworks focuses on the aspects of stability and consistency of management operations.

We then discussed the issues on inconsistencies behind policy-based management systems and presented our view on policies as a means to build extensible, flexible and programmable systems. The fundamental target is to be able to come up with a system that will be able to sustain requirement changes and evolve gracefully through policies without any changes to its carefully thought-out, “hard-wired” initial logic. The relationship with mobile code paradigms and approaches used in active networks was also discussed and justified. The above characteristics of Policy-based management have never been explicitly presented in the literature and as such this constitutes a research contribution in its own right.

We proposed generic guidelines for designing policy-based systems and explored further the issue on where to draw the line between static management intelligence and logic dynamically introduced in the system through policies.

We studied the coexistence of policies with hierarchical management systems and proposed a generic framework. The issue of the refinement of a high level policy into policies enforced in every layer of the management hierarchy has been introduced and the cases when this methodology is applicable have been identified.

We proposed a generic framework for dynamically creating management services in the system through policies. The translation of policies into interpreted code, which represents the logic that combines components modelling the hardwired functionality of the system, enables the creation of services while the system is running.

Most of the work presented in this chapter has been published in [Fleg01]. The guidelines for the design of policy-based systems were mainly put together by A.K. Bandara and were briefly described in [Band05]. The above proposed approaches to policy based management are validated through their application in the context of a system for managing QoS in IP DiffServ Networks. The latter has been designed with policy extensibility in mind and is presented in detail in the next chapter. We look into providing a holistic approach for managing QoS and we identify all the required functionality from service to resource related aspects. The presentation of the QoS management architecture that follows in the next chapter focuses in the policy related issues and proposes a generic classification of QoS related policies.

Chapter 4

4 Policy-based QoS Architecture

Chapter 4 of this thesis proposes a Policy-based QoS architecture for managing IP Differentiated Services Networks. While the DiffServ QoS model specifies control and data plane mechanisms for providing QoS, there is a need for network and service management functionality, which is an integral part of QoS-based telecommunications networks. The proposed architecture is influenced by the TMN framework and addresses both service and network management issues, including offline and dynamic mechanisms. QoS management has always been one of the most popular applications of policies since every ISP can realise their own business objectives through policies in a flexible manner regarding offered services and treatment of customer traffic in their network. In this chapter, we show how policies can be applied to the proposed QoS management architecture and also propose a generic classification of QoS policies that can be enforced on the various functional components of the system.

IP Differentiated Services is widely seen as the framework to provide Quality of Service (QoS) in the Internet in a scalable fashion. However many issues have still not been fully addressed, such as the way Per-Hop Behaviours (PHBs) and Per Domain Behaviours (PDBs) can be combined to provide end-to-end services. Other key aspects not yet fully addressed are admission control, resource reservations and the role of management plane functionality and its integration with the control and data planes. The work proposed in this chapter provides an integrated architecture and associated techniques for automated QoS delivery in a DiffServ-capable IP network. Our approach deals with both service and resource management aspects and the relationships between them. We assume the existence of the basic component for defining QoS-based IP connectivity services, i.e. the Service Level Specification (SLS), which is the technical part of a Service Level Agreement (SLA). The standardisation of the set of the parameters present in SLSs is essential for the development of automation and dynamic negotiation of SLSs between customers and providers. We also define the concept of the QoS Classes (QCs) which expose the edge-to-edge network QoS capabilities and provide the "glue" between the customer specific SLSs and the router elementary PHBs.

We adopt a holistic view for providing QoS-based services: vertically, from QoS services to PHBs, and horizontally, from service request handling to service fulfilment and assurance. We propose a two-level, approach to both service management and traffic engineering. To the best of

our knowledge, none of the related works considers both service-related and traffic engineering aspects, and in addition none of them considers multiple hierarchical levels in traffic engineering or admission control.

Policy management plays an important role in our proposed QoS management architecture since it provides the ability to the administrator to guide its behaviour and eventually the behaviour of the network through high-level declarative directives that are dynamically introduced in the system. As presented in chapter 3, we actually view policies as a means of extending the functionality of management systems dynamically, in conjunction with pre-existing "hard-wired" management logic. We use the methodology also presented in the previous chapter for applying policies to the proposed architecture since the latter is a hierarchical distributed management system. We propose a generic classification of the policies for managing QoS, based on the components they apply to in our architecture. The proposed work on QoS policies differs a lot from the existing work in the literature since it covers all the policy-related aspects of QoS Management, from Service management and Traffic Engineering down to low-level device configurations. The detailed description of the QoS policies following the aforementioned classification will be presented in Chapter 5.

The structure of the chapter is as follows.

Section 4.1 presents a proposed DiffServ Layered IP Service model describing all the layers required for achieving performance guarantees, starting from the high level Service Level Agreement (SLA) down to the elemental DiffServ node functionality, i.e. the PHB. A description of the parameters present in a Service Level Specification (SLS), the technical part of an SLA, is provided as defined in the context of the European IST project TEQUILA [TEQUILA]. Then the definition of the QoS Class is presented which provides the missing link between the SLS and the PHB and expose network wide network capabilities.

Section 4.2 describes the proposed Policy-based QoS Management architecture. A high level description of the architecture is first presented, describing all the sub-systems, namely Policy Management, SLS Management, Traffic Engineering and Monitoring. Then every sub-system of the architecture is decomposed into functional blocks and their functionality and interactions are discussed in the relevant subsections (4.3-4.6). Fundamental issues of the operation of the architecture are also discussed while a working system scenario is presented in section 4.7.

Section 4.8 presents a generic classification of the QoS Policies that can be applied to our architecture based on the "position" of the component whose behaviour they influence. Section 4.9 provides an implementation view of the proposed QoS management architecture depicting all the software components and repositories as well as the technologies used for performing all the interactions and management operations in order to implement such a management system.

Finally, 4.10 summarises what has been presented in this chapter, highlighting the research contributions.

4.1 A Layered Service Model for IP QoS

The basic DiffServ QoS concept is the PHB, exposing in a generic way the router QoS capabilities. PHBs may be implemented by smart scheduling and buffer management mechanisms such as Weighted Fair Queuing (WFQ) and Priority Queuing and algorithms for implementing packet dropping policies such as Random Early Detection (RED) [Floy93]. The PHB is the basic building block for supporting value-added IP services, previously negotiated between the provider and its customers through SLAs. However there is clearly a "missing link" between the low-level data-plane concept of a PHB and a high-level IP transport service like Voice over IP (VoIP). This is illustrated in the next Figure. The two higher layers are discussed in section 4.1.1 while the lower layers are discussed in section 4.1.2.

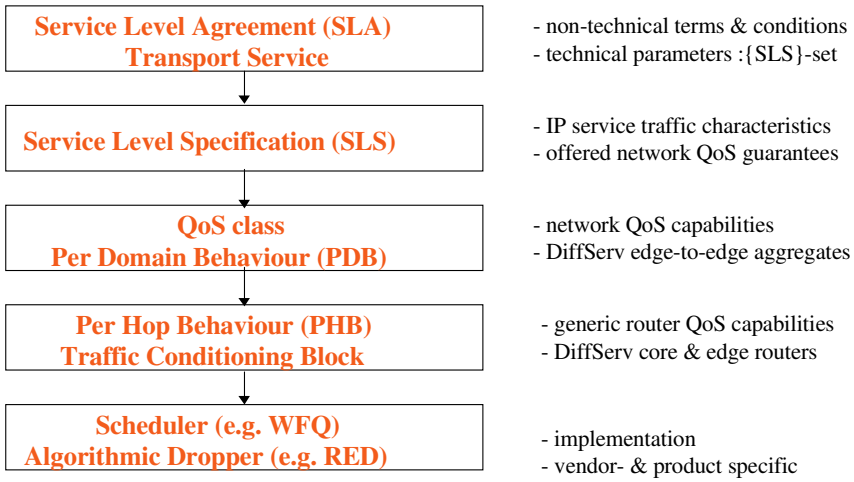


Figure 4-1: A proposal for a DiffServ Layered Service Model

4.1.1 Service Level Specifications

The two higher layers in Figure 4-1 describe the interface between the IP transport provider and its customers. According to the IETF DiffServ working group [RFC2475], a *Service Level Agreement* (SLA) is "*the documented result of a negotiation between a customer and a provider of an IP service that specifies the levels of availability, serviceability, performance, operation or other attributes of the transport service*". The SLA contains technical and non-technical terms and conditions. The technical specification of the IP connectivity service is given in *Service Level Specifications* (SLSs). An SLS "*is a set of technical parameters and their values, which together define the IP service, offered to a traffic stream by a DiffServ domain*". SLSs describe the traffic

characteristics of IP flows and the QoS guarantees offered by the network to these flows. Note that a SLA may contain a set of SLSs. Since a SLS is by definition uni-directional, the description of e.g. a bi-directional Virtual Leased Line (VLL) requires two SLSs.

The DiffServ working group of the IETF did not have the intention to further specify the content of an SLS beyond the loose definitions given above. Nevertheless the definition of an SLS is a key-step towards the provisioning of value-added IP services because it specifies the technical interface between the provider and the customer, i.e. *technical terms and conditions*. Therefore the TEQUILA project [TEQUILA] has taken the initiative to propose a standard template for the parameters and semantics of an SLS [Gode01]. In the following table, we only give the basic parameter groups of the SLS template and a brief description. More specifically an SLS has the following fields: Scope, Flow Description, Traffic Conformance Testing, Excess Treatment, Performance Parameters, Service Schedule and Reliability.

Table 4-1 SLS Parameters

Parameter Group	Description
Scope	Identifies the geographical region *where* the contract is applicable by e.g. specifying ingress and egress interfaces.
Flow description	Identifies *the packet stream* of the contract by e.g. specifying a packet filter (DSCP, IP source address, etc).
Traffic descriptor	Describes the traffic envelop through e.g. a token bucket, allowing to identify in-and out-of-profile packets
Excess Treatment	Specifies the treatment of the out-of-profile packets at the network ingress edge including dropping, shaping and re-marking.
Performance Parameters	Specifies the QoS network guarantees offered by the network to the customer for in-profile packets including delay, jitter, packet loss and throughput guarantees.
Service Schedule	Specifies *when* the contract is applicable by giving e.g. hours of the day, month, year.

The *Scope* of an SLS associated to a given service identifies uniquely the geographical and topological region over which the QoS of the IP service is to be enforced. An ingress (or egress) interface identifier should uniquely determine the boundary link or links as defined in [RFC2475] on which packets arrive/depart at the border of a DS domain. This identifier may be an IP address, but it may also be determined by a layer-two identifier in case of e.g. Ethernet, or for unnumbered

links like in e.g., PPP-access configurations. The semantics allow for the description of one-to-one (pipe), one-to-many (hose) and many-to-one (funnel) communication SLS-models, denoted respectively by (1|1), (1|N) and (N|1).

The *Flow Description* (*FlowDes*) of an SLS associated to a given service indicates for which IP packets the QoS policy for that specific service is to be enforced. An SLS has only one FlowDes, which can be formally specified by providing one or more of the following attributes:

FlowDes = (DSCP, source information, destination information, application information)

Setting one or more of the above attributes formally specifies an SLS FlowDes. The DSCP is the DiffServ Code Point in the IP header. The source/destination information could be a source/destination address, a set of them, a set of prefixes or any combination of them. The FlowDes provides the necessary information for classifying the packets at a DS boundary node. The packet classification can either be Behaviour Aggregate (BA) or Multi-Field (MF) based.

Traffic Envelope and Traffic Conformance describe the traffic characteristics of the IP packet stream identified by FlowDes. The traffic envelope is a set of Traffic Conformance (TC) parameters, describing how the packet stream should be in order to receive the treatment indicated by the *Performance Parameters* (see below). The TC parameters are the input to the *Traffic Conformance Testing* algorithms. The traffic conformance testing is the set of actions, which uniquely identifies the "in-profile" and "out-of profile"[2] (or excess) packets of an IP stream identified by the Flow-ID. The TC Parameters describe the reference values the traffic identified by the FlowDes will have to comply with. The TC Algorithm is the mechanism enabling to unambiguously identify all "in" or "out" of profile packets based on these conformance parameters. The following is a non-exhaustive list of potential conformance parameters: *peak rate p* in bits per sec (bps), *token bucket rate r* (bps), *bucket depth b* (bytes), *minimum MTU* - Maximum Transfer Unit - *m* (bytes) and *maximum MTU M* (bytes).

An *Excess Treatment* parameter describes how the service provider will process excess traffic, i.e. out-of-profile traffic (or other than the in-profile in the case of multi-level TC). The process takes place after Traffic Conformance Testing. Excess traffic may be dropped, shaped and/or remarked. Depending on the particular treatment, more parameters may be required, e.g. the DSCP value in case of re-marking or the shapers buffer size for shaping.

The *Performance Parameters* describe the service guarantees the network offers to the customer for the packet stream described by the FlowDes and over the geographical/topological extent given by the scope. There are four performance parameters: *delay*, *jitter*, *packet loss, and*

[2] Note that the conformance result might not necessarily be of a binary mode (in/out) but it could also be multi-level (e.g. using a Two-rate Three-colour Marker algorithm).

throughput.[3] *Delay* and *jitter* indicate respectively the maximum packet transfer delay and packet transfer delay variation from ingress to egress. Delay and jitter may either be specified as worst-case (deterministic) bounds or as quantiles. The *packet loss* indicates the loss probability for in-profile packets from ingress to egress. Delay, jitter and packet loss apply only to in-profile traffic. *Throughput* is the rate measured at the egress.

Performance parameters might be either quantitative or qualitative. A performance parameter is quantifiably guaranteed if an upper bound is specified. The service guarantee offered by the SLS is quantitative if at least one of the four performance parameters is quantified. If none of the SLS performance parameters is quantified, then the performance parameters *delay* and *packet loss* may be "qualified". Possible qualitative values for delay and/or loss are *high, medium, low*. The actual "quantification" of the relative difference between high, medium and low is a policy-based decision (e.g. high = 2 x medium; medium = 3 x low). If the performance parameters are not quantified nor qualified the service will be best effort.

The *Service Schedule* indicates the start time and end time of the service, i.e. when is the service available. This might be expressed as a collection of the following parameters: time of the day range, day of the week range, and month of the year range. *Reliability* indicates the maximum allowed mean downtime per year (MDT) and the maximum allowed time to repair (TTR) in case of service breakdown. Other parameters might be also included in the SLS for example the *Assurance Level*, which describes the percentage of the time by which the ISP will be able to conform to the other SLS parameters.

4.1.2 QoS Classes

The third layer in Figure 4-1 is the "network QoS layer" in-between the customer-specific SLS-based services and the elementary PHBs supported by the routers. The notion of the QoS Class (QC) is introduced to substantiate this mediation. QoS classes expose network-wide QoS transport capabilities and they are bound to the specific technologies deployed and capabilities provided by the network. For example, a Virtual Wire (VW) QoS Class could be defined to denote an edge-to-edge transport capability with a guaranteed maximum packet delay and a guaranteed throughput for an aggregate IP packet stream marked as Expedited Forwarding (EF). QoS Classes should be seen as specifications of a Per-Domain-Behaviour [RFC3086]. We have adopted the following specification of QoS Class shown in Table 4-2.

[3] For each of these parameters we must specify a *time interval* and in some cases (e.g. delay) a quantile.

Table 4-2: The formal definition of a DiffServ QoS class

Parameter	Description
Ordered Aggregate	The allowed values are: Expedited Forwarding (EF), Assured Forwarding 1-4 (AF1, AF2, AF3, AF4), Best Effort (BE)
Delay	The *delay* is the maximum *edge-to-edge* delay that the in-profile packets of a certain IP stream should experience. It is a continuous parameter that may be worst case (deterministic) or a percentile (probabilistic).
Packet Loss	The *packet loss* is the upper bound of the *edge-to-edge* packet loss probability that in-profile profile packets of a certain IP stream should have.

A finite number of QoS-Classes is obtained by allowing only a discrete number of possible delay and loss values. The delay-loss ranges are mainly driven by the corresponding performance parameters of the services offered (expressed in the SLSs) and they are subject to the capabilities and characteristics of the network including its topology. Furthermore, they may be policy-influenced, changing from time to time as service and network policies warrant so.

A network supports certain QoS classes by implementing dedicated PHBs in the core routers, appropriate Traffic Conditioning Blocks at the edge routers and an overall resource management system. Supporting customer specific SLSs then boils down to a "service mapping" of the SLS onto a QoS class and SLS admission control while the network should be suitably engineered to gracefully sustain the traffic of the admitted SLSs.

4.2 A Policy-Based Quality of Service Management Architecture

In order to support end-to-end quality of service based on Service Level Subscriptions (SLSs), besides the data plane functionality of DiffServ Per Hop Behaviour (PHB) a need for management plane functionality has also been identified through the notion of Bandwidth Broker as presented in Chapter 2. While its basic functionality has been defined, its architecture remains largely unspecified. Therefore we substantiate and extend the notion of the BB by defining the functionality of an integrated management and control architecture that combines both service negotiation/invocation and traffic engineering aspects. This architecture can be seen as a detailed decomposition of the concept of BB realized as a hierarchical, logically and physically distributed system instead of the centralised model proposed by IETF. All the components of the system are policy-driven enabling the administrator to drive its behaviour through high-level directives. In

the rest of the section we present a high level description of the architecture, highlighting its basic features.

Since our major goal is to try to provide services to customers with QoS guarantees similar to telecommunication services such as VoIP, our architecture is heavily inspired by the TMN framework presented in Chapter 3. It comprises the following main parts starting from the higher level as shown in Figure 4-2: Policy Management, SLS Management, Traffic Engineering, Monitoring and Data Plane functions. Although in the figure we present our system following the B-ISDN reference model [BISDN] categorised by management, control and data plane functionality, the functionality can be grouped based on the logically layered architecture described by the TMN model. Policy management corresponds to the business management layer, SLS management to the Service Management layer, Traffic Engineering and Monitoring to the Network and Element Management Layer and finally the data plane functionality to the Network Element layer. Although in Figure 4-2 the SLS Management and Traffic Engineering can be seen as management systems of the same layer, SLS management belongs conceptually to a layer above the Traffic Engineering in the management hierarchy. The distinction between network and element management is not clear in this abstract view of the architecture but the sub-systems can be decomposed in components with network wide view as well as components responsible for managing individual network elements as described in the following sections.

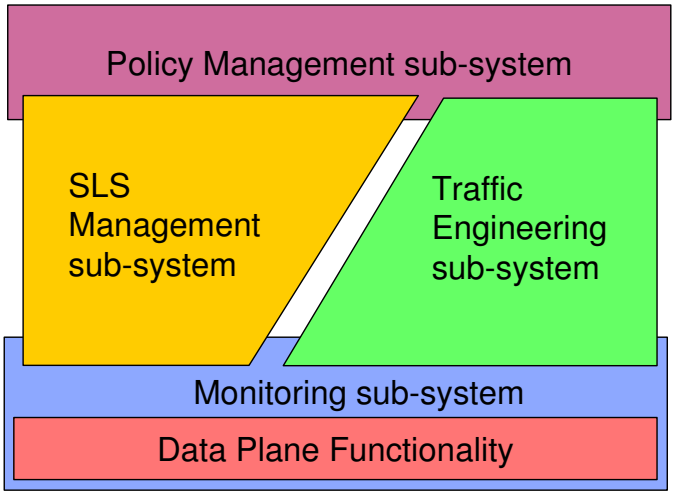


Figure 4-2 High-level view of the Policy-based QoS Management Architecture

Policy Management plays an important role in this architecture since it enables the administrator to drive the behaviour of the other parts of the architecture and eventually the network behaviour based on his/her business objectives. It comprises all the necessary components for creating, storing and enforcing policies in an automated fashion, leading to a more flexible and extensible system. This means that every operator that deploys our proposed system can introduce policies dynamically in a high-level declarative manner and these are dynamically evaluated without the

need for system re-engineering. The latter is very important in the QoS management domain since multi-service provisioning is a process highly based on business objectives and policies in order to provide the means to address the continuously changing requirements of the operator.

The SLS Management part is responsible for subscribing and negotiating SLSs with users or other peer Autonomous Systems (ASs) and it performs admission control for the dynamic invocation of subscribed SLSs. This part is also responsible for transforming the SLS specific information into aggregate traffic demand (traffic matrix), in order to *feed* the Traffic Engineering part with the necessary input.

The Traffic Engineering part is responsible for selecting paths which are capable of meeting the QoS requirements for a given traffic demand. Such information is conveyed between the customer and the service provider during SLS negotiation, it is then processed by the Traffic Forecast and is transformed into the aggregate traffic matrix. The TE part of the architecture is responsible for dimensioning the network according to the projected demands, and for establishing and dynamically maintaining the network configuration that has been selected to meet the SLS demands.

The Monitoring part is responsible for providing the other parts of the architecture with the appropriate network measurements. In addition, it is responsible for assuring that the contracted services are indeed delivered at their specified QoS.

Finally, the Data Plane functionality includes the PHBs and Traffic Conditioning Blocks that need to be present in the routers of the managed network for providing the elementary DIffServ capabilities as defined by [RFC2475].

All the above sub-systems do not act in isolation. Their functionality is decomposed and described in the following sections together with their interactions with the rest of the components of the system.

4.3 SLS Management

SLS Management is responsible for all SLS-related activities and is further decomposed into three Functional Blocks (FBs): SLS Subscription, SLS Invocation, and Traffic Forecast. Figure 4-3 shows the interaction of the SLS Management components with external customers or ISPs and the rest of the subsystems in the architecture.

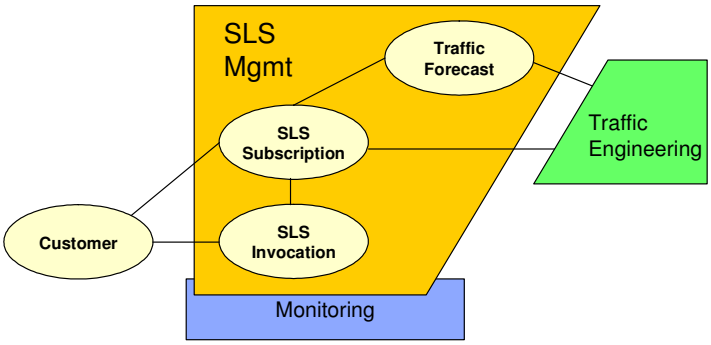


Figure 4-3 SLS Management decomposition

4.3.1 Two-level SLS Management

We distinguish two epochs for service requests: *subscription* and *invocation*. At subscription epochs, customers subscribe to services in order to gain the right to use these later. At invocation epochs, customers invoke services to which they have subscribed. Services can be invoked either *explicitly*, using signalling to the edge of the network or *implicitly*, being active throughout the subscribed service schedule (e.g. leased-line, Virtual Private Network services). On successful service invocation, the corresponding traffic flows are generated in the customers' hosts and injected to the network.

The distinction between service subscription and invocation is due to a number of reasons. It directly corresponds to current business models and practices, it is required for AAA (Authentication, Authorization and Accounting), it enables business-driven service provisioning, facilitates traffic demand derivation and prompts for higher aggregation levels at customer levels, therefore increasing scalability. We include admission logic at both subscription and invocation epochs, with the purpose not to overwhelm the network with traffic beyond its current capacity.

4.3.2 SLS Subscription

SLS Subscription (SLS-S) includes processes of customer registration and long-term policy-based admission. The customer might either be a peer Autonomous System (AS) or a business or residential user. The subscription (or registration) concerns the Service Level Agreement (SLA), containing amongst other prices, terms and conditions and the technical parameters of the SLS. The subscription should provide the required *authentication information* for Authentication, Authorisation & Accounting (AAA) purposes when an SLS will be eventually invoked. SLS-S contains an SLS repository with the current (long-term) subscriptions and a SLS history repository. This information serves as basic input for the Traffic Forecast. SLS-S performs static "admission control" in the sense that it knows whether a requested long-term SLS can be supported or not given the current network configuration; this is not an instantaneous snapshot of

load/spare capacity, but a longer-term configuration provided by Network Dimensioning (described below).

4.3.3 SLS Invocation

SLS Invocation (SLS-I) is the FB, which includes the process of dynamically dealing with a flow and it is part of *control plane* functionality. It performs dynamic admission control as requested by the user and, as such, it can be flow-based. SLS-I receives input from the SLS-S, e.g. for authentication purposes, and has a view on the current spare resources. Admission control is mostly measurement-based and takes place at the network edges. Finally, SLS-I delegates the necessary rules to the traffic conditioner. The rules when enforced will ensure that packets are marked with the correct DSCP, so that out-of-profile packets are handled in a certain way, etc.

4.3.4 Traffic Forecast

The main function of *Traffic Forecast* (TF) is to generate a traffic estimation matrix to be used by the TE sub-system. Traffic Forecast is the "glue" between the SLS Management Customer-oriented Framework and the TE Resource-oriented Framework of our functional architecture. The *input* of TF is *SLS (customer) aware* while the *output* is only *QoS Class aware*. The *traffic estimation matrix* contains *per QC type*, the (long-term) estimated traffic that flows between each ingress/egress pair. Its calculation is based on the SLS subscription repository, traffic projections and historical data provided by Monitoring, network physical topology, the physical nature and capacities of the access links, business policies, economic models, etc. A *first level of aggregation* is to bundle all the intra-domain traffic and as such a source-destination matrix will be made for each QC type. A *second level of aggregation* takes also into account inter-domain, transient traffic and it combines all traffic that needs to end at a certain destination AS.

4.4 Traffic Engineering

In general, there exist two TE approaches:

MPLS-based TE: This approach relies on an explicitly routed paradigm, whereby a set of routes (paths) is computed off-line for specific types of traffic. In addition, appropriate network resources (e.g. bandwidth) may be provisioned along the routes according to predicted traffic requirements. Traffic is dynamically routed within the established sets of routes according to network state.

IP-based TE: This approach relies on a 'liberal' routing strategy, whereby routes are computed in a distributed manner, as discovered by the routers themselves. Although route selection is

performed in a distributed fashion, the QoS-based routing decisions are constrained according to network-wide TE considerations made by the dimensioning and dynamic routing algorithms. The latter dynamically assigns cost metrics to each network interface. Route computation is usually based on shortest or widest path algorithms with respect to the assigned link costs. In order to allow for routes to be computed per traffic type or class, a link may be allocated multiple costs, one per DSCP.

In this thesis, we are considering only the MPLS-based approach, although our architecture is independent of particular TE approach, i.e. it can be used to accommodate also pure IP-based TE solutions.

4.4.1 Two-level Traffic Engineering

We propose a two-level traffic engineering system operating both at long-to-medium and medium-to-short time scales (Figure 4-4). At the long-to-medium time scale, Network Dimensioning maps the traffic requirements to the physical network resources for providing dimensioning directives in order to accommodate the predicted traffic demands. At the medium-to-short time scales, we manage the routing processes in the network (Dynamic Route Management), performing dynamic load balancing over multiple edge-to-edge paths, and we ensure that link capacity is appropriately distributed among the PHBs (Dynamic Resource Management) in each link by appropriately selecting the scheduling discipline and buffer management parameters.

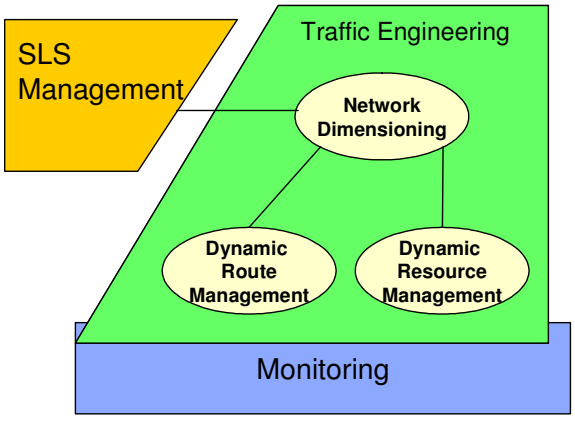


Figure 4-4 Traffic Engineering sub-system decomposition

The above approach for service-driven Traffic Engineering can be also referred to as *first plan, then take care*, whereby decisions, taken with long-term perspectives, are refined by dynamic functions according to actual developments in shorter time periods. The initial decisions and their refinements are driven by clearly defined objectives to meet QoS requirements and optimise network performance. This approach is not monolithic, as pure centralised schemes, while at the

same time it harnesses the dynamics of state/load-dependent schemes on the basis of guidelines produced with a longer-term vision. Thus, the provisioning of the network is effectively achieved by both taking into account the long-term service level subscriptions in a time dependent manner and the dynamic network conditions that are state dependent.

We argue that both levels are necessary when considering an effective traffic engineering solution, since independently have shortfalls (*centralised* vs. *distributed*, *time-* vs. *state-dependent*, *static* vs. *dynamic*), which can only be overcome when they are used in conjunction.

4.4.2 Network Dimensioning

Network Dimensioning (ND) is responsible for mapping the traffic onto the physical network resources and configures the network in order to accommodate the forecasted traffic demands. ND defines MPLS Label Switched Paths (LSPs) in order to accommodate the expected traffic. The TF FB provides the forecasted demand and ND is responsible for determining cost-effective allocation of physical network resources subject to resource restrictions, load trends, requirements of QoS and policy directives and constraints. The resources that need to be allocated are mainly QoS routing constraints, such as link capacities and router buffer space, while the means for allocating these resources are capacity allocation, routing mechanisms, scheduling, and buffer management schemes. The ND component is centralised for a particular AS, although distributed implementations on a sub-domain or area of an AS are also possible. In any case, it utilises network-wide information, received from the network routers and/or other functional components through polling and/or asynchronous events.

The main task of ND, which operates in the order of days to weeks[4], is to accept input about the forecasted demand from TF, and by knowing the physical topology to calculate, in a policy-driven fashion, and install parameters required by the elementary TE functions in the IP routers. The output of ND is the set of LSPs and their associated parameters. The objective of such a calculation is to accommodate all the expected demand, and therefore meet the SLS performance requirements, without overloading any part of the network. This objective leaves space for unpredictable traffic fluctuations (handled by Dynamic Route Management and Dynamic Resource Management) and at the same time not having to reroute large amounts of traffic in the case of failures. One can devise ND algorithms either in the form of an optimisation problem and use relaxation techniques to overcome the complexity problems or by using heuristics.

[4] By which we mean that it is invoked at approximately these intervals - not that the algorithms take this long to converge! For example, as presented in [Trim03b], for a 300-node network, the average running time of the proposed algorithm is 17 min.

The output of ND is fed to Dynamic Route Management (DRtM) and Dynamic Resource Management (DRsM), and also to the SLS Management part of the architecture in order to base the admission control decisions for future SLS subscriptions. Admission control for SLS invocations is based on the information from ND, DRtM and DRsM, with the latter two being more important since they have more up-to-date dynamic information.

4.4.3 Dynamic Route Management

Dynamic Route Management (DRtM) is responsible for managing the routing processes in the network according to the guidelines produced by ND on routing traffic according to QoS requirements associated to such traffic (contracted SLSs).

This component is responsible mainly for managing the parameters based on which the selection of one of the established LSPs is effected in the network, with the purpose of load balancing. It receives as input the set of LSPs (multiple LSPs per source-destination pair) defined by ND and relies on appropriate network state updates distributed by DRsM. In addition, it informs ND, by sending notifications, on over-utilisation of the defined paths so that appropriate actions are taken (e.g. creation of new paths). In this approach, the functionality of the DRtM is distributed at the network border routers/edges.

In MPLS-based TE the LSP bandwidth is *implicitly* allocated through link scheduling parameters along the topology of the LSPs, while traffic conditioning enforced at an ingress router is used to ensure that input traffic is within its defined capacity.

4.4.4 Dynamic Resource Management

Dynamic Resource Management (DRsM) has distributed functionality, with an instance attached to each router. This component aims at ensuring that link capacity is appropriately distributed between the PHBs sharing the link. It does this by setting buffer and scheduling parameters according to ND directives, constraints and rules and taking into account actual experienced load as compared to required (predicted) resources. Additionally DRsM attempts to resolve any resource contention that may be experienced while enforcing different PHBs. It does this at a higher level than the scheduling algorithms located in the routers themselves.

DRsM gets estimates of the *required* resources for each PHB from ND, and it is allowed to dynamically manage resource reservations within certain constraints, which are also defined by ND. For example, the constraints may indicate the *effective* resources required to accommodate a certain quantity of unexpected dynamic SLS invocations. Compared to ND, DRsM operates on a relatively short time-scale. DRsM manages two main resources: Link Bandwidth and Buffer Space.

Link Bandwidth: ND determines the bandwidth required on a link to meet the QoS requirements conveyed in the SLS. DRsM translates this information into scheduling parameters, which are then used to configure link schedulers in the routers. These parameters are subsequently managed dynamically, according to actual load conditions, to resolve conflicts for physical link bandwidth and avoid starving of such bandwidth for the enforcement of some PHBs.

Buffer Space: Appropriate management of the buffer space allows packet loss probabilities to be controlled. The buffers also provide a bound on the largest delay that successfully transmitted packets may experience. Buffer allocation schemes in the router dictate how buffer space is split between contending flows and when packets are dropped. According to the constraints imposed by ND for the QoS parameters associated with the traffic of a given PHB, DRsM sets the buffer space and determines the rules for packet dropping in the routers. The drop levels need to be managed as the traffic mix and volume changes. For example, altering the bandwidth allocated to a LSP may alter the bandwidth allocated for the correct enforcement of a corresponding PHB. If the loss probability for the PHB is to remain constant, then the allocated buffer space may need to change.

DRsM also triggers ND when network/traffic conditions are such that its algorithms are no longer able to operate effectively. For example, link partitioning is causing lower priority/best effort traffic to be throttled due to excessive high priority traffic and these conditions cannot be resolved within the constraints previously defined by ND.

4.5 Monitoring

The state-dependent dynamic traffic engineering functions require the observation of the state of the network through a monitoring system in order to apply control actions to drive the network to a desired state. A monitoring system should provide information for the following three categories of tasks:

- Assist traffic engineering in making provisioning decisions for optimising the use of network resources according to short to medium term changes (near-real time monitoring).
- Assist traffic engineering in providing analysed traffic and performance information for long-term planning in order to optimise network usage and avoid undesirable conditions.
- Verify whether the QoS performance guarantees committed in SLSs are in fact being met.

We view traffic engineering as a continual and iterative process of network performance improvement. The optimisation objectives may change over time as new requirements and

policies are imposed, so the monitoring system must be generic enough to cope with such changes.

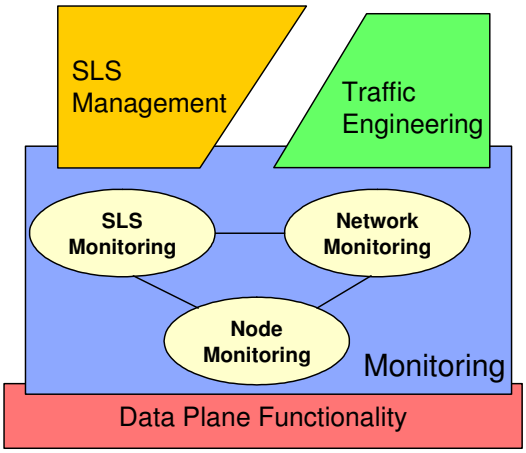


Figure 4-5 Monitoring sub-system decomposition

Monitoring large-scale traffic engineered networks is a difficult task and requires mechanisms for data collection from a variety of network nodes, aggregation of heterogeneous data sets, data mining of large data sets and data analysis to generate results for providing feedback to other functional subsystems that require monitoring information. We designed a hierarchical monitoring system to cooperate with the rest of our subsystems, which has the following components:

Node Monitor is responsible for node related measurements and there exists one Node Monitor per router. It is able to perform active measurements between itself and any other Node Monitors, at path or hop level, as well as passive monitoring of the router it is attached to. Node Monitor collects measurement results from meters or probes located at routers through *passive* or *active* monitoring agents.

Network Monitor is responsible for network-wide post-processing of measurement data using a library of statistical functions. It is centralised and utilises network-wide performance and traffic measurements collected by all the Node Monitor entities in order to build a physical and logical network view (i.e., the view of the routes, i.e. LSPs, that have been established over the network).

Service Monitor is responsible for customer related service monitoring, auditing and reporting. SLS Monitor is centralised, since it must keep track of the compliance of the level of service provided to customer SLSs. It utilises information provided by Network Monitoring and the various Node Monitors. SLS Monitor handles the requests for activation or deactivation of monitoring a particular set of SLSs.

4.6 Policy Management

Policy Management includes components such as the Policy Management Tool (PMT), Policy Repository (PR), and the Policy Consumers (PCs) or Policy Decision Points (PDPs). A single PMT exists for providing a policy creation environment to the administrator where policies are defined in a high-level declarative language and after validation and static conflict detection tests, they are translated into object-oriented representation (information objects) and stored in the repository. PR is a logically centralized component but may be physically distributed since the most widely adopted technology for implementing this component is the LDAP (Lightweight Directory Access Protocol) Directory [LDAP]. After the policies are stored, activation information may be passed to the responsible PC/PDP in order to retrieve and enforce them.

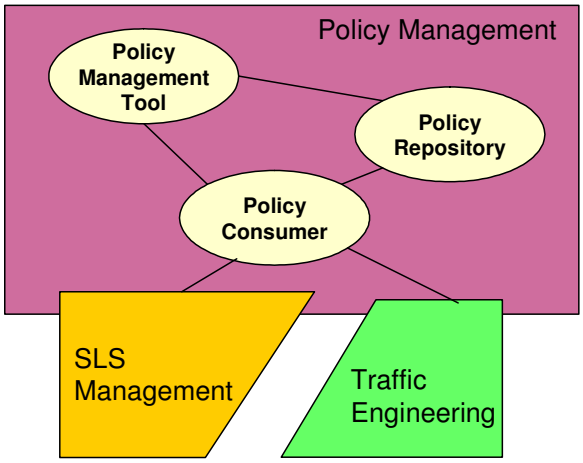


Figure 4-6 Policy Management sub-system decomposition

The methodology for applying policies to a hierarchically distributed system, such as our architecture, was described in Chapter 3. Although Figure 4-6 shows a single PC/PDP over all the functional blocks for illustrative purposes, our model assumes many instances of policy consumers. Every instance of the policy consumer is attached to the component that is influenced by the policies this PC is responsible to enforce (tightly coupled approach), complementing in this way the static management intelligence of the above layer of the hierarchy. For example, a policy enforced on the DRsM component is actually enhanced management logic that conceptually belongs to the ND layer of our model. Policies may be introduced at every layer of our system but higher-level policies may possibly result in the introduction of related policies at lower levels, mirroring the system hierarchy as discussed in the previous chapter. A concrete example of the previous case will be presented in the next chapter where we present specific policies the can be enforced in our architecture.

The above tightly coupled approach introduces a decentralisation of policy management functionality in our system since policy management tasks are delegated to the policy consumers

which are distributed to every component mirroring the physical distribution of the component they manage. This of course reduces significantly the amount of remote interactions in our system and consequently minimises the management overhead in terms of bandwidth consumption. For example, having a single policy consumer managing all the DRsM components will lead to many interactions with the latter since they are distributed in every node of the network. Moreover, enforcement of most of the policy rules result in operations with the managed component, both for monitoring i.e. evaluating the conditions, and configuration i.e. executing the actions.

The delegated management tasks in our system are represented by the policies defined by the administrator which of course are constrained to the types of policies conceived at the requirements engineering phase of the system design and are translated in interpreted code which is dynamically executed. As discussed in the previous chapter, this approach is similar to the Management by Delegation paradigm but instead of distributing code representing management tasks, policies in a "constrained" object-oriented format are distributed, reducing significantly the security risks. The above distribution of management tasks (policies) is done in a scalable fashion since policies are not defined in one by one fashion for every component but for example a single policy rule can be defined that applies to all the DRsM components in the system. The distribution of policies is automatically taken care by the policy system i.e. the policy management tool notifies the responsible policy consumers when a new policy becomes available and these go and retrieve the relevant policies from the policy repository.

4.7 The Big Picture

In Figure 4-7, the policy-based QoS management architecture is illustrated, showing all the individual components of every sub-system as presented in the previous sections.

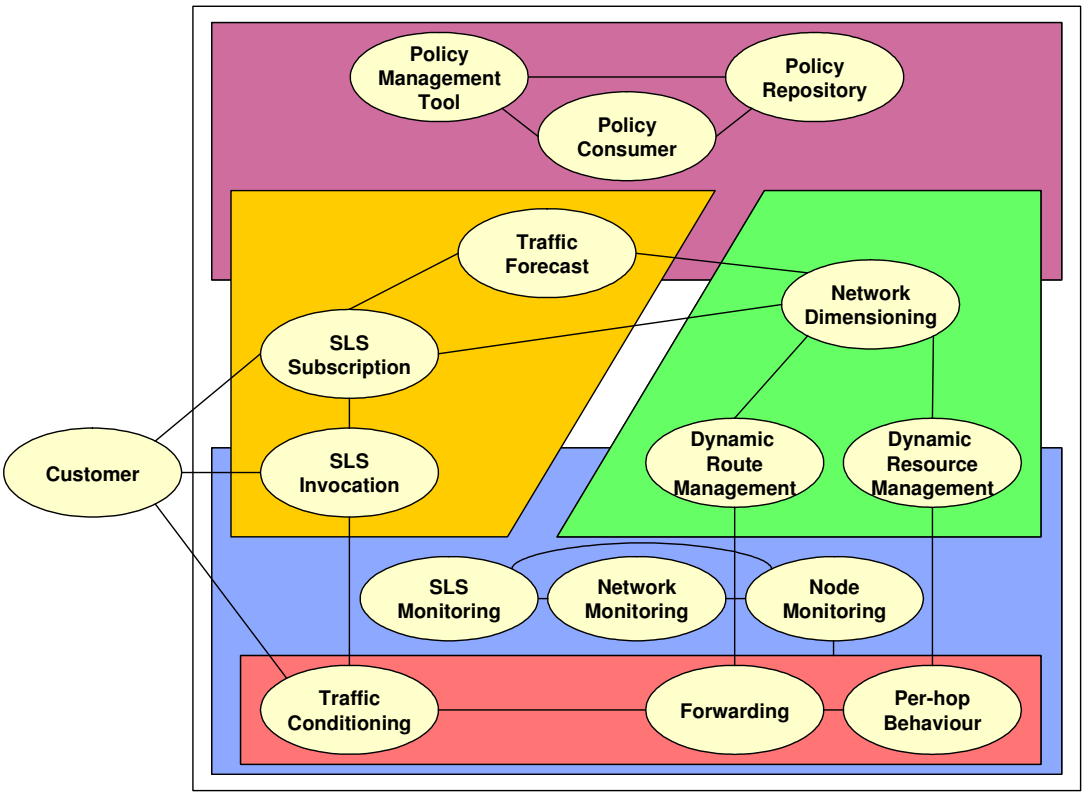


Figure 4-7 Detail view of the Policy-based QoS Management architecture

We describe below some fundamental issues regarding the operation of the architecture, presenting also a working system scenario describing the information flow between the various components.

4.7.1 Resource Provisioning Cycle - RPC

As scalability and convergence are of primary concern, interactions and information exchanged between service layer and traffic engineering functions are not fine-grained but they rely on aggregate information in order to avoid oscillations and high overhead. As such, the service layer and traffic engineering functions exchange information (traffic and resource availability matrices), which refers to *Traffic Trunks* and not individual customer contracts or flows, at the granularity of *Resource Provisioning Cycles (RPCs).*

A Resource Provisioning Cycle is a period of time during which anticipated QoS traffic demand, for the supported traffic trunks, is believed to be valid. Should anticipated QoS traffic demand prove no longer valid, a new RPC is initiated. New RPCs may also be initiated when the service layer or the Traffic Engineering functions realise, each from its own perspective, that they can no longer gracefully provide the QoS being requested. Traffic Engineering functions realize that, when the network can no longer deliver the QoS targets of the supported QoS Classes, and service

layer functions when they can no longer satisfactorily sustain the actual offered load at the contracted QoS levels. Note that RPCs may also be initiated by network administration (e.g. when new points of presence or resources are installed).

The interactions between the service layer and the Traffic Engineering functions occur only at RPC epochs, not at the granularity of every single (or a few) service requests. As such, only when a new RPC commences a traffic matrix is produced, the network is appropriately engineered and the Resource Availability Matrix (RAM) is produced. By their definition, RPCs are relatively long time periods, ranging from days to weeks. RPCs are bound to (macro level) traffic forecasts that are usually drawn with long-term perspectives.

4.7.2 Dynamic Service Management and Traffic Engineering functions

As anticipated QoS traffic demand and network availability estimates are forecasts, they should be treated by the Traffic Engineering and service layer functions as such. Actual offered traffic is to fluctuate around the forecasted values, some times even beyond acceptable statistical errors. Our architecture interprets QoS traffic forecasts as nominal rather than actual values and network availability interprets them as guidelines, rather than stringent directives. On these grounds, the system incorporates *dynamic service layer and Traffic Engineering functions,* which, by utilizing actual network state and load information, try to optimise network performance in terms of service admissions and resource utilization while meeting at the same time traffic QoS constraints.

4.7.3 Working System Scenario

In this section, we will describe a working scenario and the information flow of the functional architecture that was presented in the previous sections.

Let's assume that several customers are attached to an AS which employs the proposed architecture. These customers are negotiating SLSs with the SLS-S FB. Let's assume that at some point in time there are *N* subscribed SLSs, and at time *t* re-dimensioning needs to be done. The reasons for re-dimensioning might be: (a) the amount of spare resources for future SLS subscriptions is below a (policy defined) threshold, (b) the amount of the SLS subscription rejections is greater than a (policy-based) defined threshold, (c) DRtM or DRsM are unable to handle the current resource demand, (d) the re-dimensioning cycle has elapsed (dimensioning period).

First, ND will request the traffic forecast (matrix) that corresponds to the next dimensioning period. TF will consider the currently subscribed *N* SLSs, the (policy-based) additional *M* SLS subscription requests that are predicted for the next dimensioning period, the (policy-based) over-

subscription ratio, and historical monitoring data, in order to prepare the traffic matrix[5]. The demand provided to ND will be something between (*N*, *N+M*). ND will run some optimisation or heuristic dimensioning algorithm in order to define multiple paths (trees) between the ingress and (list of) egress nodes, i.e. the configuration of the network for the next dimensioning period. ND needs to provide this configuration information back to SLS-S in order to be able to perform admission control at the level of subscriptions. This information is also passed to DRtM and DRsM, which contact the Network Elements (NEs) in order to enforce this configuration by setting LSPs and configuring the various PHBs. Finally, Monitoring needs to be informed about this configuration in order to initialise the appropriate monitoring engines.

The SLS-S will use the configuration received from ND to decide for future subscriptions but will also pass it to SLS-I in order for it to have the necessary information for invocation admission control. Now lets assume that several SLSs are being invoked. For each of these SLSs the SLS-I will check the SLS repository to see if it corresponds to a subscribed customer. The current load information (taken from monitoring) will also be checked against the current network configuration in order to decide whether a particular SLS can be accepted or not. If it is accepted, SLS-I will configure the Traffic Conditioners (data plane) appropriately. When the actual traffic arrives, DRtM will balance the load among the multiple existing paths. If there are many SLSs invoked, it might be the case that more resources are required because of the over-subscription ratio. Then DRsM and DRtM will try to find more resources, but always within the ND's guidelines. If this procedure is not successful there are two alternatives: either the invocation is not accepted, if the situation occurred before the admission request; or, re-dimensioning is invoked, if the problem happened after admission as a result of many ingress nodes receiving simultaneous admission requests.

Policy management influences almost all of the parts of the previous scenario. A more concrete example is the following. If there is an administrator's policy according to which 10% of the overall network resources should always be available to best effort traffic, then ND needs to take that policy in mind during the calculation of the configuration. In addition, DRsM needs to be aware of this policy so that it does not allow dynamic requests for additional resources corresponding to other QC to reduce this percentage of resources for best effort traffic.

4.8 Classification of QoS Policies

The components in the architecture can be categorised logically according to which part they belong: SLS components on the left side and TE components on the right (separated horizontally)

[5] There are actually more than one such matrices, one per QoS Class (QC)

and hierarchically depending on which layer of the hierarchy they belong (separated vertically). The Traffic Forecast component has an SLS aware part, which belongs to the SLS Management part, and an SLS unaware part that belongs to the TE part. Components that are located in the lower level i.e. SLS-I, DRtM and DRsM are dynamic, measurement-based while those that reside in the upper level i.e. ND and SLS-S are static, time-based and off-line.

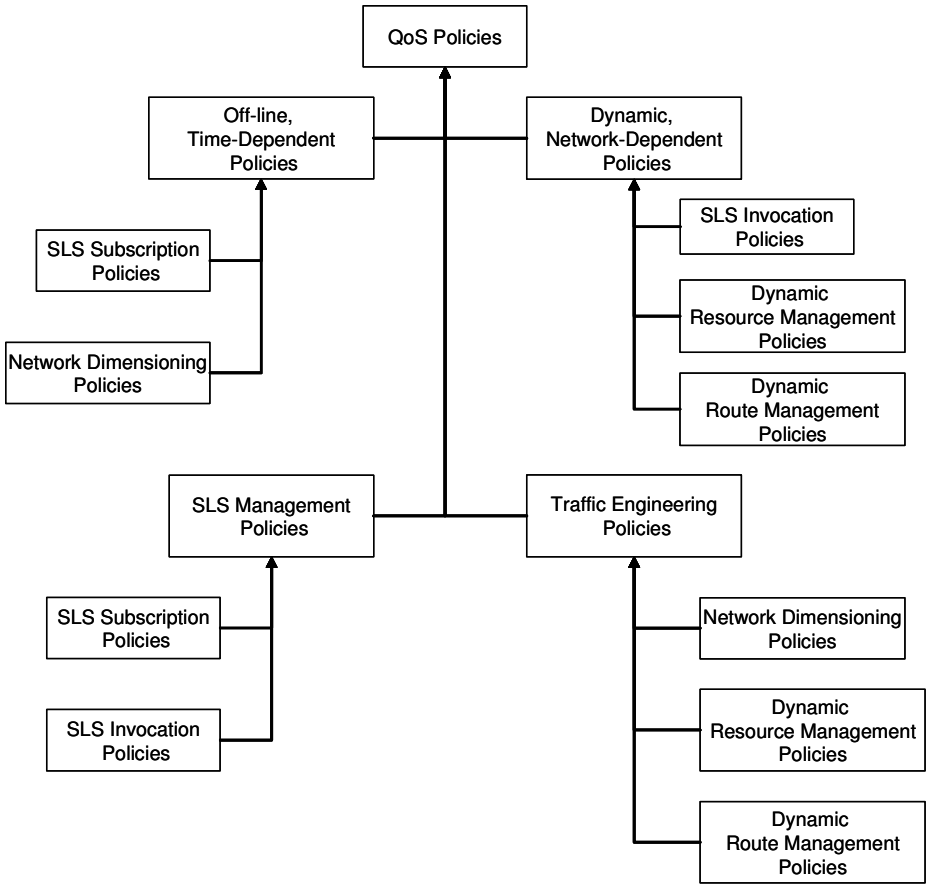


Figure 4-8 Classification of QoS Policies

The same classification can be applied to policies enforced in our system, based on the "position" of the component they relate to (Figure 4-8). So, SLS Management policies are those enforced to the SLS-S and SLS-I components and they are related mostly to directives and constraints regarding admission control decisions while TE policies are those enforced to the ND, DRsM and DRtM components regarding allocation of resources and routing processes. Categorisation of policies can also be done according to the layer of the hierarchy the policy-influenced component belongs. Policies that apply to the upper layer of the hierarchy i.e. SLS-S and ND are more time dependent and the trigger for their execution depends mostly on the state of the component they apply to. On the other hand, policies enforced to the lower level of the hierarchy i.e. SLS-I, DRtM and DRsM are more measurement-based, meaning that they are triggered not only by the component they influence but also by certain conditions and events in the network. Such network

state dependent triggers are registered by the correspondent policy consumers to a Monitoring component. The latter notifies them when an event has occurred so that the relevant policy consumer is triggered to enforce the policy actions.

Another classification of QoS policies applied to our architecture can be done according to the granularity of information they refer to based on the service model for IP DiffServ presented in section 4.1. Policies applied to SLS management can dictate the behaviour of the relevant components with regard to SLSs and customer related information while policies applied to the TE part of the architecture can only influence the behaviour of the latter with regard to treatment of QoS Classes. Policies that define customer specific treatment cannot be applied to the Traffic Engineering components of our system since the latter only deals with aggregated SLS information represented by different QoS classes. Consequently policies that apply to the core routers of the network can only influence the different treatment of the supported PHBs as dictated by the DiffServ model.

4.9 Management System Architecture

In this section, we describe an example implementation of the functional policy-based QoS architecture presented in this chapter [Trim03], focusing on the technologies and protocols used to realise such a management system.

The architecture of the management system that realises the described functionality in terms of applications and their interactions is depicted in Figure 4-9. The Service Manager (SM) performs admission control for service subscriptions, holds the Service Repository (SR) and configures edge routers with SLS related information. The Network Dimensioner (ND) gets the traffic matrix from SM and performs off-line traffic engineering, producing the logical network configuration and configuring network nodes. Both SM and ND use the network topology kept in the Network Repository (NR) although they have different views of it: SM has a service-centric view of edge nodes only while ND has a complete view of the physical network topology. The network topology can be modelled in CIM (Common Information Model) [CIM] and implemented through an LDAP (Lightweight Directory Access Protocol) database. ND updates NR with the logical topology every time the network is re-dimensioned. Both SM and ND are policy-driven through the Policy Management Tool (PMT) that provides a graphical policy interface to the human network/service operator. Note that the functionality of Policy Consumers is integrated in the SM and ND applications for enforcing the relevant policies.

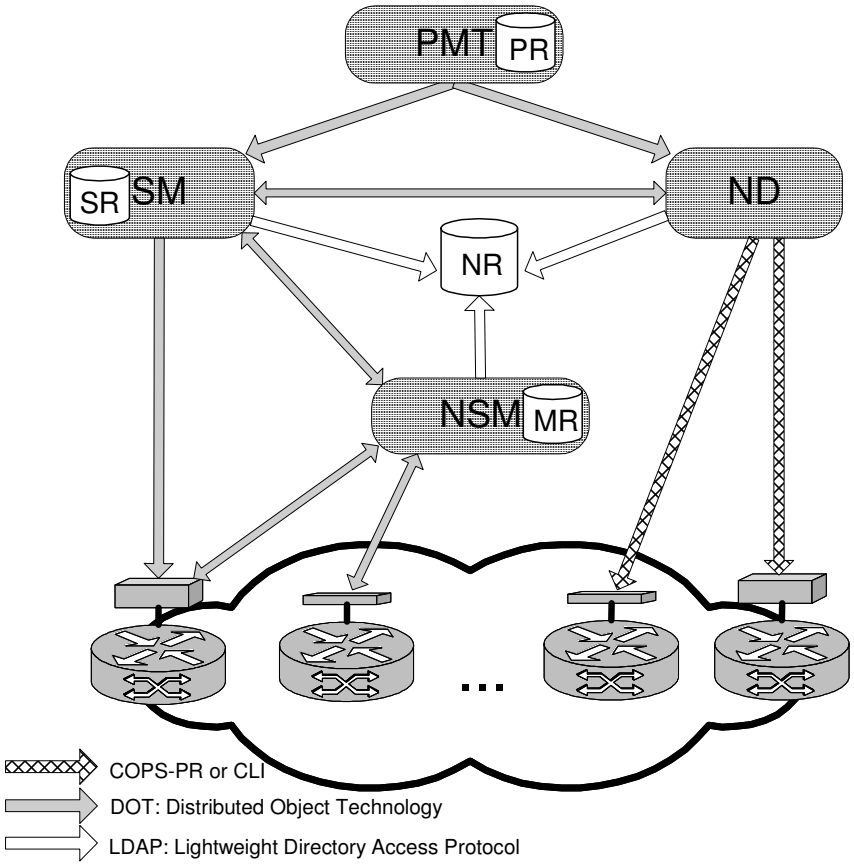


Figure 4-9 Management implementation architecture overview

The Network and Service Monitor (NSM) is a centralized application that configures monitoring activities at each node and subsequently receives and stores events in the Monitoring Repository (MR). NSM uses the logical and physical network topology for its monitoring tasks. The interactions with the NR are LDAP-based [RFC2251] while the interactions between any of the other management components are based on Distributed Object Technology (DOT). A CORBA (Common Object Request Broker Architecture) [CORBA] platform has been used in the current implementation but the interface modelling is general enough to be used with emerging Web-based DOTs such as SOAP (Simple Object Access Protocol)/XML [Pavlou04]. The configuration interactions to the network are based on COPS-PR (Common Open Policy Service for Provisioning) [RFC3084]] wherever possible while DOT interfaces have been used for functionality not yet supported through COPS PIBs (Policy Information Bases) [RFC3318].

Every network node consists of the actual Label Switched Router (LSR) and a computing node attached to it that hosts node monitoring, admission control and route management functionality which is not currently supported by DiffServ-capable LSRs (the computing nodes are depicted over the switching nodes in the figure). ND configures scheduling and buffering parameters in all nodes through the DiffServ COPS PIB [RFC3317] while it passes path information to edge nodes

through telnet-CLI (Command Line Interface); the latter then set-up the edge-to-edge paths through signalling. SM also configures edge nodes with SLS-related data, including conditioning for SLS traffic and associated source addresses. This is also done through CORBA since relevant PIBs do not exist. Finally, NSM configures node monitors through CORBA while SNMP is used for local monitoring, between the computing station and the LSR.

In summary, we have used existing technologies such as LDAP, COPS-PR, CLI and SNMP as much as possible while we resorted to more flexible DOTs where relevant standards were absent. The additional functionality DiffServ-capable LSRs should have in order to support our architecture is the following: Admission Control at invocation level in order to accept flows conforming to SLSs, flows outside a SLS envelope and flows without a SLS at all; and Route Management in edge nodes for load balancing among LSPs supporting a traffic trunk.

4.10 Summary and conclusions

In this chapter, we described the salient characteristics of a Policy-based QoS Management architecture of IP DiffServ networks. The proposed architecture provides a holistic approach to intra-domain QoS, including both service management and traffic engineering functionality. All the sub-systems comprising the architecture were presented and decomposed into functional blocks providing detailed descriptions of their functionality and interactions with the rest of the components of the architecture. The management plane aspects of our architecture include SLS subscription, traffic forecasting, network dimensioning and dynamic resource & route management. Most of these are policy-driven. The control plane aspects include SLS invocation and packet routing while data plane aspects include traffic conditioning and PHB-based forwarding. The management plane aspects of our architecture can be thought as a detailed decomposition of the BB concept in the context of an integrated management and control architecture that aims to support both qualitative and quantitative QoS-based services.

Policy Management plays an important role in the architecture since it provides the ability to the administrator to guide dynamically the behaviour of various components of the architecture based on the business objectives of the operator. Since our architecture is a hierarchical distributed system, we applied the methodology presented in Chapter 3 for enforcing policies. We then presented a generic categorisation of QoS policies applied to our architecture depending on the "position" of the components they influence. The above classification differs a lot from the QoS policies identified by other researchers and the IETF, which are limited to policies configuring the edge routers of a DiffServ/IntServ network. Our work provides a holistic view of QoS policies starting from SLS Management policies, network wide resource management policies down to

router management policies. The detailed description of the QoS policies as well as their formal representation is presented in the next chapter.

Finally, a system implementation view of our architecture was presented describing all the implementation technologies that can be used in order to realise the proposed management system on top of a real DiffServ capable IP network. A more detailed description is provided in Chapter 6 where we present our implementation of the policy management system demonstrating also the enforcement of example QoS policies presented in the next chapter.

The work presented in this chapter is the result of collaboration within the TEQUILA project [TEQUILA]. The design of the proposed architecture was put together by the whole project consortium [Trim01] while the author of this thesis drove the policy management aspects of the architecture. The contributions related to the design of the policy sub-system, the application of policies to the architecture as well as the generic categorisation of QoS policies presented are the author's alone [Fleg02]. Most of the components of the architecture were validated in the context of the project and results were presented in [Trim03].

Chapter 5

5 QoS Policies: Service Management and Traffic Engineering Policies

Chapter 5 of this thesis defines policies that are related to QoS Management of multi-service IP networks driving the behaviour not only of the routers of the managed network but also of the components of the management system presented in the previous chapter. We follow the generic categorisation of the QoS policies presented in section 4.8, trying to provide a holistic approach to the area of policies for QoS Management, which has not been addressed in the literature up to now. We go further and refine every category of QoS policies presented, identifying the parameters that are influenced by policies addressing both Service Management and Traffic Engineering aspects of QoS Management for both offline and dynamic components. For the formal representation of policies we have adopted an object oriented representation based on the Policy Core Information Model (PCIM) and its extensions (PCIMe) as jointly defined by the Policy Framework working group of IETF and DMTF and we propose relevant extensions to represent the proposed QoS policies. Finally, using as a case study the proposed QoS policies, we validate the methodology for applying policies to hierarchical management systems as well as the management service creation framework presented in Chapter 3.

QoS Management has always been one of the most popular applications of Policy-based Management since it requires a more sophisticated management paradigm due to the increased complexity of the managed network with the corresponding QoS mechanisms. Most of the related work in the literature has focused on policies configuring edge devices of the network reflecting the relevant SLSs but has failed to address any issues related to service management and network-wide resource management of QoS enabled IP networks. In our work, we provide a holistic view to QoS policies addressing all the aspects of QoS management categorised by the relevant components of the architecture they apply to as presented in the previous chapter. Service Management policies are identified mostly for driving the behaviour of SLS admission control both at the subscription as well as at the invocation level. Traffic Engineering related policies enforce the decisions of the administrator on how to distribute resources to the different service classes both at a static level based on predicted traffic demand, as well as in a dynamic manner based on the actual network status. A good understanding of the functionality of every component

of the architecture was required in order to identify the parameters and the corresponding management logic that is left to be defined by policies. Although the policies presented in this chapter are based on the functionality and algorithms developed in the context of the EU IST TEQUILA project, most of them are quite generic and could apply to any QoS management system.

A formal representation of the policies is essential in every policy-based system in order to bridge the gap between the human policy administrator who defines the policies and the actual enforcement commands executed at the component in order to realize the business goals of the administrator. This also facilitates the consistency checks that should take place since policies are introduced in the system dynamically as discussed in Chapter 3. The widely adopted PCIM information model together with its extensions defined jointly by IEFT and DMTF was adopted to represent the QoS policies presented in this chapter since it provides the core classes for the specification of policies for any application. In this chapter, we propose the required extensions to these information models in order to represent the proposed Service management and Traffic Engineering policies by extending the basic classes of PCIM and PCIMe.

The methodology for applying policies to a hierarchical management system as well as the generic framework for creating management services almost exclusively through policies are validated by presenting specific case studies from the QoS policies defined in this chapter. More specifically, the relationship between the policies driving the behaviour of the static offline traffic engineering component, i.e. network dimensioning (ND), with the dynamic resource management (DRsM) policies applied to every router interface is thoroughly examined. We also discuss the case where a ND policy could possibly result in the introduction of a DRsM policy, presenting a specific example. Finally, we investigate the case of introducing the functionality of the DRsM component almost exclusively through policies in our system, by combining static hardwired functionality with the dynamically introduced policy logic. For the above case we present a series of policy rules needed to be defined in order to introduce such functionality.

The structure of this chapter is as follows.

Sections 5.1 and 5.2 define the policies that influence the components of the SLS Management and Traffic Engineering sub-systems of the Policy-based QoS architecture presented in the previous chapter. The sections are structured as follows. First, a summary of the functionality and algorithms of every component is presented, followed by a description of the relevant policies that influence their behaviour. Relevant examples are given through out these sections using our policy language, which is presented in the next chapter. Finally, the extensions to PCIM and PCIMe required to represent the previously defined policies are depicted by UML class hierarchy diagrams together with a description of the relevant classes.

Section 5.3 uses as a case study the Traffic Engineering policies applied to the ND and DRsM components to provide a proof of concept validation of the methodology for applying policies to hierarchical management systems. The two components were chosen due to their hierarchical relationship. Section 5.4 presents another case study again for validation purposes of the theoretical framework presented in chapter 3 for creating management services almost exclusively through policies. The DRsM component was chosen for this purpose presenting examples of series of policies that could be applied in order to instantiate a DRsM service in our system using policies.

Finally, section 5.6 summarises what has been presented in this chapter, highlighting the research contributions.

5.1 SLS Management policies

5.1.1 System Architecture

In Figure 5-1 we present only the SLS management part of the architecture presented in Chapter 4 together with the components of the policy management sub-system i.e. Policy Management Tool, Policy Repository and Policy Consumer needed to make the system extensible through policies. As shown, policy consumers are attached to the components they manage (tightly-coupled approach) for both the centralised and the distributed components i.e. SLS subscription management and SLS Invocation management respectively. A single policy consumer is needed for driving the behaviour of the SLS-S component while policy consumers are distributed in every edge router of the managed network enforcing locally SLS-I policies. Based on the classification of the policies driving the behaviour of the QoS management architecture (section 4.8), we show the interactions of the policy consumers with the monitoring components required for receiving policy triggering events which depend on the network state.

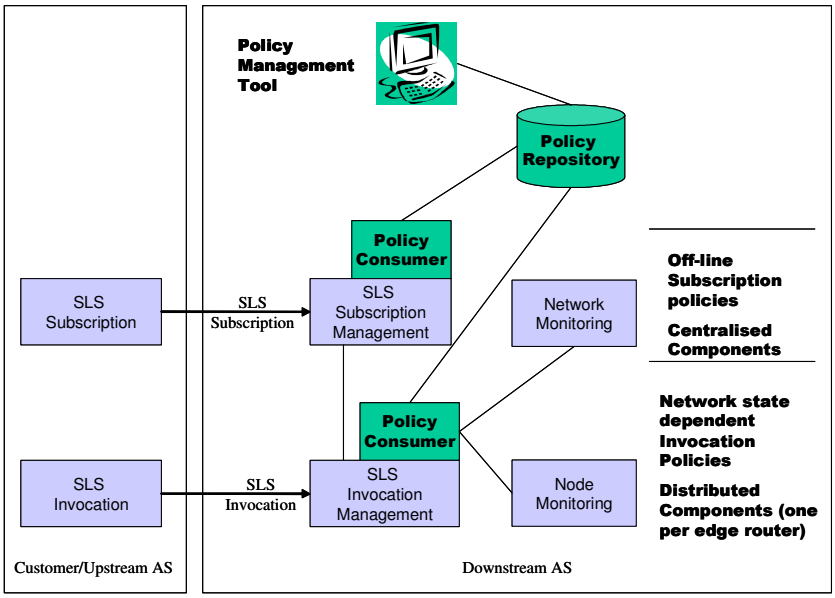


Figure 5-1 Policy-driven SLS Management System Architecture

Given the fact that SLS-S is an offline component, the corresponding policy consumer does not need to communicate with any other external component since policies are triggered based on internal events, e.g. an arrival of a subscription request. On the other hand, policy consumers that drive the behaviour of the SLS Invocation components need to communicate both with network monitoring in order to receive events regarding the status of the network, e.g. if congestion occurs, and the node monitoring for getting information on the usage of the currently invoked services. The behaviour of the components described here is based on the work presented in [Myko03]. The SLS management policies are further explained in the sections below.

5.1.2 SLS Subscription Management

5.1.2.1 Functionality Description

In this section, we provide a description of the behaviour of the SLS Subscription management module. The latter is responsible for deciding whether to accept or not a SLS subscription request in order not to eventually overwhelm the network while at the same time maximise the number of subscriptions. It also provides the logic behind the negotiation process, i.e. proposing alternatives to subscribers that their subscriptions could not be accepted. The functionality of the SLS-S module can be decomposed into 3 distinct categories described in more detail below.

SLS traffic forecast. From the relevant parameters of each requested SLS, it deduces the expected traffic based on traffic demand forecast factors. This traffic is then aggregated with the expected traffic accumulated from the SLSs established during this Resource Provisioning Cycle (RPC).

Admission Logic. The resulting aggregating traffic is mapped against the corresponding entries of the resource availability matrix (RAM). The result of this mapping is fed to the admission control algorithm, which will determine whether this request should be accepted or rejected, in the latter case because the risk of overwhelming the network with traffic that cannot be served with the guaranteed QoS is too high. In case the request is rejected, the upper SLS traffic limit the network can sustain is deduced.

Counter-Offer Calculator. This component is responsible for calculating the parameters of the alternative SLSs to the SLSs requested by the customer that could not be accepted. These parameters are calculated based on the rejected SLS and the acceptable traffic limits deduced by the admission logic. These alternative SLSs are proposed as counter offers to the customers.

5.1.2.2 SLS subscription Management Policies

In this section, we describe in detail the functionality of the admission control and counter-offer calculator presented above focusing on the parameters and decisions that can be defined by policies and we present specific examples of such policies guiding the behaviour of the SLS-S module. In the table below, we provide a summary of the SLS subscription policies described in this section, describing the policy actions and the parameters based on which policy conditions are formed.

Table 5-1 Summary of SLS Subscription Policies

SLS Subscription Policies	
Policy Actions	**Policy Conditions**
Admission Logic Actions - Accept SLS - Reject SLS **Counter-Offer Actions** - Downgrade Availability - Downgrade Qos Class - Downgrade Service Rate -Reschedule	**Level of Anticipated Traffic Demand with respect to ranges in RAM** $(0..R^{a}_{min})$, $(R^{a}_{min}..R^{w}_{min})$, $(R^{w}_{min}..R_{max})$ **SLS parameters** - Customer ID - Performance Parameters: QoS Class - Requested Service Rate - Scope of the SLS: Ingress, Egress

The *admission logic* algorithm decisions are based on the Resource Availability Matrix (RAM) calculated by the Traffic Engineering sub-system and especially by the offline component i.e.

Network Dimensioning. The RAM provides an availability estimate per traffic trunk (TT), which is expressed in the form of a resource availability buffer (RAB) shown in Figure 5-2. R^{a}_{min} represents the available bandwidth for this TT guaranteed by the network at any time. The R^{w}_{min} represents the available bandwidth for this TT guaranteed by the network at congestion times. This bandwidth is not hard-reserved by the network and it can be utilised by other TTs but when congestion occurs the TE system will force all TTs to be constrained to their R^{w}_{min} bandwidth limit. Finally, R_{max} represents the maximum available bandwidth for this TT but with no guarantees. The anticipated traffic demand of the newly requested SLS aggregated with the demand of already subscribed SLSs as calculated by the SLS traffic forecast is compared against the RAM in order to take the decision whether to accept the request or not. The level in the RAM up to which the administrator will allow subscriptions is left to be defined by policies, reflecting the ISP's business objectives. The trade-off is evident, the more subscriptions you accept thus increasing your profit the lower guarantees you offer to your customers. Consequently, the behaviour of the admission logic algorithm depends entirely whether the company (ISP) will target many and not so demanding customers or fewer with higher demands.

Figure 5-2 Resource Availability Buffer

These business objectives are realised through the policies the administrator can define and influence the behaviour of the algorithm. Different behaviour of the algorithm can be specified depending on the QoS class or even on the TT that the SLS corresponds to by defining the acceptable region in the RAB. The buffer up to R^{w}_{min} is considered that can be used with high confidence levels because the network has been engineered so that this buffer can be at least available even in congestion times. The confidence levels are obviously subject to multiplexing errors inherent in the calculation of the traffic demand and availability estimates of the current RPC. The buffer from R^{w}_{min} to R_{max} is considered more risky because the network cannot provide any guarantees for the rates in between this buffer due to the fact that the possibility of other TTs that share the same physical resources with the TT under consideration increases. Admission control decisions can be also based on the customer that requests a SLS subscription, for example an operator might decide to reject a SLS despite that fact that there is spare capacity in the network to accommodate its traffic due to its business relations with that specific customer.

Example 5.1 **if** SLS_Sub_QoS_Class == EF **and** Anticipated_Demand < R^{w}_{min} **then** accept_SLS

The above policy example states that SLS subscriptions for traffic that belongs to the EF QoS Class should be accepted if the anticipated demand is less than R^{w}_{min}. A policy that would lead to accept more SLSs but with low confidence levels could state to accept the SLSs for which the anticipated demand is in a level between R^{w}_{min} and R_{max}. Finally, policies for a QoS class with no guarantees like best effort could state that all subscription requests should be accepted.

The last category of policies that influence the behaviour of the Subscription management component is the negotiation logic policies. These policies are triggered every time the admission logic rejects a requested subscription. The purpose is to negotiate with the subscriber various alternatives until an agreement can be reached. Different alternatives can be proposed depending on the SLS characteristics and can be proposed, either in "all in one" mode or in "one by one" mode. The policies the administrator specifies determine the behaviour of the negotiation logic, giving the flexibility to define different alternatives. A type of alternative could be to propose the service availability guarantees which will be calculated as the ratio of the anticipated demand over R_{max}, which reflects the confidence level of the network to ensure the QoS at the requested traffic rate. Another type of alternative could be to downgrade the quality of the requested SLS, which entails proposing a different QoS class to be used to serve the requested traffic as specified by the business decisions of the operator. Moreover, an alternative for proposing to the customer would be to downgrade the requested service rate to a level that can be accepted by the network, given the current status of the anticipated demand with regard to the RAM. The proposed service could be presented as a percentage of the original request so as to ensure that the proposed alternative would indeed be accepted by the admission logic. Finally, the negotiation logic policies could propose the alternative of rescheduling, which entails to propose another "service schedule" specified in the SLS template, in the near future, with a combination of the previous alternatives if needed (e.g. downgrading the quality or the service rate etc).

Example 5.2 **if** SLS_Sub_QoS_Class == EF **and** Anticipated_Demand > R^{w}_{min} **then** propose SLS_Sub_QoS_Class == AF1

The above policy rule states that for SLS subscriptions of EF traffic if the anticipated demand is above R^{w}_{min} then AF1 QoS Class should be proposed as an alternative to the customer that requested the SLS subscription.

5.1.3 SLS Invocation Admission Control

5.1.3.1 Functionality Description

SLS Invocation management regulates the traffic entering the network. This task encompasses two different aspects: (1) Control the number and the type of the active services and (2) Control the volume and the type of the traffic injected by the active services.

The main objectives of the invocation management are:

- To maximize the number of admitted services and the QoS they enjoy, thus maximizing network utilization while preventing QoS degradation caused by overloading the network
- To effectively resolve eventual congestion
- To provide fair treatment to services network-wide

The logic of SLS-I is based on the feedback model driven by monitoring events. These monitoring events are in the form of edge-to-edge network status alarms and threshold crossing events on aggregate traffic injected into the network by the local edge. Network status is expressed by a binary flag per TT, indicating whether the delay and loss requirements associated with the corresponding QoS class are (close to) being violated or not; the network status per TT is said to be *red* or *green* respectively. On red status the invocation admission logic has to react immediately to resolve the congestion.

Locally injected traffic provided by monitoring is projected to the resource availability buffer per TT provided by Network Dimensioning to estimate the risk in QoS deterioration as a result of admitting traffic. Since congestion cannot be accurately predicted, no hard limits are imposed in the utilized resources. Instead, when the injected traffic is considered to reach a critical point, admission strategies need to proactively protect the network and the users conforming to their contractual traffic levels from greedy users tending to exhaust the network resources. Policies introduced by the administrator define the desired behaviour of SLS-I with respect to which DiffServ traffic volume regulating mechanisms and admission control strategies to be enforced based on the monitoring events received. These policies are described in detail in the following section.

5.1.3.2 SLS Invocation Management Policies

As mentioned in the previous section, the major objective is to control the traffic injected to the network in order to resolve eventual congestion. A summary of the SLS Invocation management policies defined in this is section is shown in the table below, describing the policy actions and the parameters based on which policy conditions are formed.

Table 5-2 Summary of SLS Invocation Policies

SLS Invocation Policies	
Policy Actions	**Policy Conditions**
Adjust Service Rate - Reduce rate to a % of the contractual service rate **Adjust QoS** **-** Downgrade or upgrade the QoS (remarking) **Adjust Admission Control** - Set Minimum Threshold - Set Maximum Threshold	**Level of Injected Traffic Rate with respect to ranges in RAM** $(0..R^{a}_{min})$, $(R^{a}_{min}..R^{w}_{min})$, $(R^{w}_{min}..R_{max})$ **Network Status (Green or Red)** **SLS parameters** - Customer ID - Performance Parameters: QoS Class - Requested Service Rate - Scope of the SLS: Ingress, Egress

There are three types of strategies (policy actions) the administrator could enforce to perform SLS Invocation Management: Adjust Service Rate, Adjust Quality and Adjust Admission Control.

Adjust Service Rate is the equivalent of DiffServ policing. The configuration parameters to determine the profile the traffic will be policed against are expressed in terms of almost satisfied service rates and fully satisfied service rates. Fully satisfied service rate denotes the traffic rate, which if offered to an SLS, it enjoys QoS at its contractual rate while almost satisfied service rate denotes the rate which if offered to an SLS it enjoys QoS at a rate lower than its contractual rate but above what is considered acceptable for the particular type of service. The values of the above parameters are left to be decided by the administrator depending on the type of SLS according to the ISP's business objectives.

Adjust Quality corresponds to DiffServ remarking. Adjust Quality actions are usually performed to qualitative services, downgrading or upgrading the quality they receive in terms of the QoS-Class they are serviced by.

Admission control of explicit invocations is based on a RED congestion avoidance logic [Floy93]. As in RED congestion avoidance, invocations are rejected with a probability, which varies depending on the injected traffic rate. The rejection probability is maximum at a traffic rate maximum threshold, 0 at a traffic rate minimum threshold and 1 above the maximum threshold. Minimum and maximum thresholds are set within the Resource Availability Buffer (RAB) depicted in Figure 5-2. Adjust Admission Control boils down to adjusting the abovementioned

parameters with respect to the values of R^a_{min}, R^w_{min}, R_{max} as defined by ND. The latter enables the administrator to define the behaviour of the admission control per TT regarding the likelihood of overwhelming the network in the same manner as it is achieved for the subscription admission control described in the previous section.

When the network status is green i.e. there is no congestion in the network, policies aiming to maximise service satisfaction while avoiding building congestion should be enforced. If the traffic injected in the network belonging to a particular TT reaches a level that is considered to be critical, i.e. the risk of overwhelming the network is considered to be high, then proactive actions of mild severity, in terms of the penalty the users experience, could be performed so as to deal with demanding active services that may potentially cause congestion. The critical level might vary depending on the QoS class the TT belongs to and is specified with regard to the different ranges in the RAB. The area from R^a_{min} to R_{max}, based on the guidelines from the TE system, is the area of shared resources. Each TT may eventually use all the resources within this area. R^w_{min} determines the resources the network is configured to provide in case of congestion, with no overlaps between different TTs. Therefore, the use of resources up to this limit is considered safe. The area between R^w_{min} and R_{max} represents the area where TTs competing for the same resources may use but with a certain risk, since overlapping with other TTs may happen. Different policy actions with increasing severity might be enforced as the offered load is building up within the limits of the critical area defined by the administrator.

Example 5.3 `if SLS_Inv_QoS_Class == EF and Network_Status == Green and Injected_Traffic_Rate >` R^w_{min} `then AC_min_Threshold ==` R^w_{min} `and AC_max_Threshold ==` R_{max}

The above policy example states that for SLS Invocations of traffic that belongs to the EF QoS class, if the network status is green and the injected traffic rate is above a critical level as defined by the administrator i.e. R^w_{min} then actions such as setting the minimum and maximum threshold values of the admission control algorithm to R^w_{min} and R_{max} respectively are applied. Alternative actions could be applied to adjust the service rate of the SLSs to a percentage of the actual contractual value could be performed. As mentioned before, since there is no congestion in the network taking actions that impose severe penalties to the services is not recommended. There is also a possibility for the traffic rate to exceed the value of R_{max} and yet for congestion not to occur. This is because the scheduling algorithms applied in the core network are such, that minimum throughput is guaranteed, while a single queue is allowed to use all the physical capacity, given there is no demand from other sources. The policy based approach we followed for the SLS invocation management enables the administrator to dynamically specify policies that define the behaviour of the module for services that belong to different QoS class or even different TT. An operator's policy might state that for the premium services no penalties should

be imposed and they should enjoy their contractual QoS (while the network is in the green state), despite that fact that traffic rate reaches a limit which might considered to be risky.

When the network monitoring component issues an alarm that the network is congested, policies should be enforced that aim to resolve congestion as soon as possible. These policies include actions of increased severity, enforcing penalties to active services such as policing the traffic rate to their almost satisfied rate (a fraction of the contractual rate as defined by the administrator) and by stopping admitting new flows in the network. Resolving congestion is guaranteed thanks to the operation of the TE system, which will instruct scheduling algorithms in the core network to provide R^{w}_{min} at minimum in case of congestion. For these resources there is no overlap between TTs and therefore it is guaranteed that congestion resolving logic will converge when all TTs reach this value. TTs using resources more than R^{w}_{min} should converge to this value as soon as possible. A convergence action polices the active services to this fraction of their service rate that will cause the aggregated injected traffic rate to drop to R^{w}_{min}. When the network state becomes green again, policies should be enforced that target for a smooth and effective mediation, avoiding oscillations in network status. These policies should relax some of the severe actions executed when the network was under congestion until the network is considered to be at a safe level where again the previously defined strategies will be enforced.

Example 5.4 `if SLS_Inv_QoS_Class == EF and Network_Status == Red and Injected_Traffic_Rate > R^w_min then Adjusted_Rate == 80% and AC_max_Threshold == 0`

The above policy rule enforces severe penalties for SLS Invocations of traffic that belongs to the EF QoS Class, in the case the network is congested and the injected traffic rate is above R^{w}_{min}. These penalties are realised by adjusting the invoked service rate to 80% of the contractual rate and by stopping admitting new flows i.e. setting the maximum threshold of admission control to 0.

5.1.4 PCIM Extensions for SLS Management policies

This section proposes an object oriented information model for representing the SLS Management policies presented in the previous sections based on the IETF Policy Core Information Model (PCIM) and its extensions (PCIMe). The SLS Management Policy Information Model class inheritance hierarchy is rooted in PCIM and PCIMe.

5.1.4.1 SLS Management Policy Actions

SLS Management policy actions are modelled using classes, which are derived by the abstract PolicyAction class defined in PCIM. First, the SLS Subscription Management policy actions are

described below. The class inheritance hierarchy for both the SLS Subscription and Invocation policy actions is depicted below.

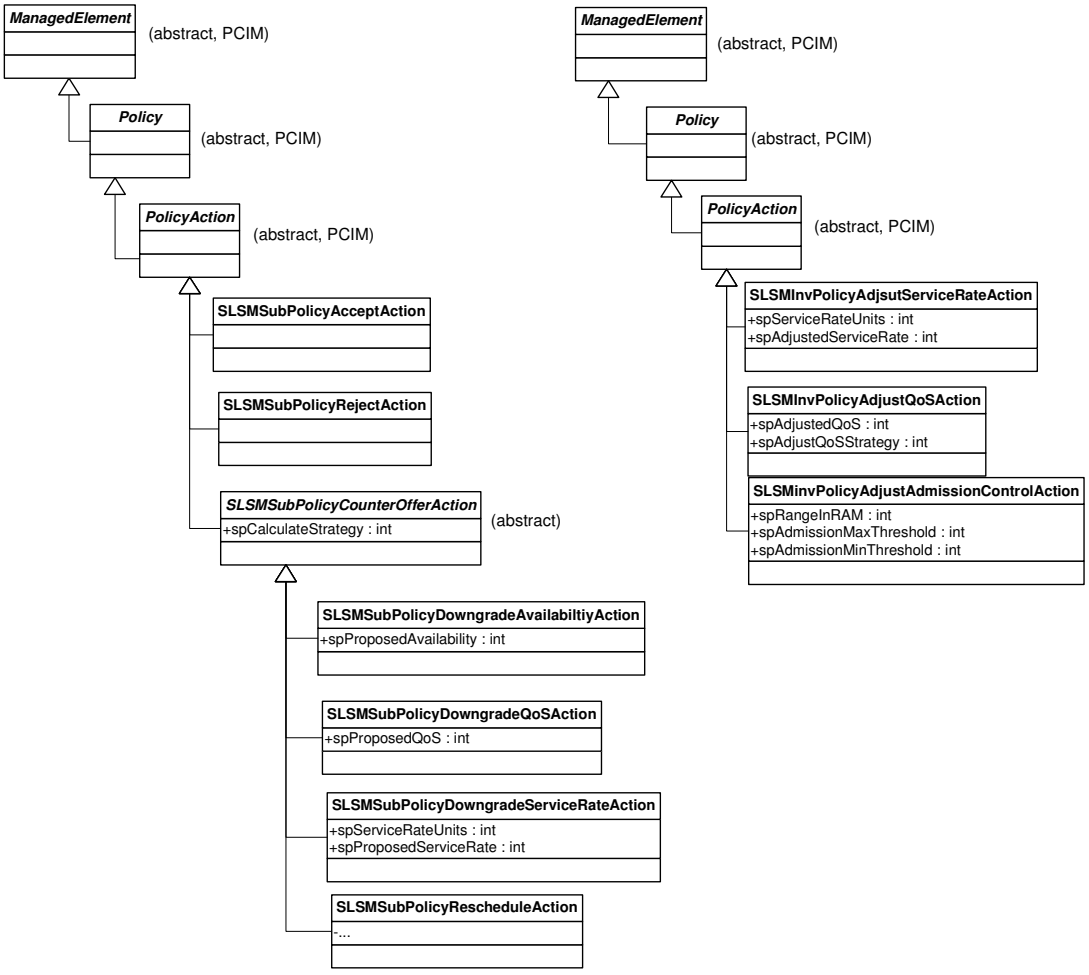


Figure 5-3 SLS Management Policy Actions – Class Inheritance Hierarchy

5.1.4.1.1 SLS Subscription Policy Actions

Since we provide the logic of the subscription admission control using policies, the policy actions that implement this decision are either accept or reject the requested SLS, represented by the classes SLSMSubPolicyAcceptAction and SLSMSubPolicyRejectAction respectively. The enforcement of these actions will be based on the conditions of the policy rules which can depend on the requested SLS parameters as well as the aggregated traffic demand projected on the RAM as described previously. The next group of actions that can be enforced to the subscription management component are those that represent the counter offer policy actions. The latter actions are all derived from an abstract class SLSMSubPolicyCounterOfferAction, which contains a property spCalculateStrategy. This property is an enumerated integer that represents predefined

methods for calculating the value of the counter-offer SLS parameter i.e. Service Availability, QoS Class, Schedule, Service Rate.

The classes SLSMSubPolicyDowngradeAvailabilityAction, SLSMSubPolicyDowngradeQoSAction, SLSMSubPolicyDowngradeServiceRateAction, SLSMSubPolicyRescheduleAction represent the specific counter offer actions. When the spCalculateStrategy is set to 0, then an absolute value should be provided for the revised SLS parameter to be proposed as an alternative, which is included in every counter-offer action class. An example of a different strategy represented by a different value of this property might be to determine the QoS Class with the most available bandwidth to be offered when the corresponding counter offer action is enforced. The spProposedAvailability property of the SLSMSubPolicyDowngradeAvailabilityAction represents the absolute value of the alternative availability value in % and the spProposedQoSClass property of the SLSMSubPolicyDowngradeQoSAction is an enumerated integer representing the predefined QoS Classes supported in the provider's domain. The spProposedServiceRate property of the SLSMSubPolicyDowngradeServiceRateAction defines the alternative service rate of the requested SLS and can be specified either in kbps or in a % of the originally proposed value depending on the value of spServiceRateUnits property. Finally, the SLSMSubPolicyRescheduleAction class contains all the necessary properties so that the administrator can specify a time period using the same properties defined in the TimePeriodCondition class defined in PCIM.

5.1.4.1.2 SLS Invocation Policy Actions

There are three classes representing the policy actions that can be applied to the SLS invocation management component. All these classes are directly derived from the abstract class PolicyAction defined in PCIM. The first policy action class is the SLSMInvPolicyAdjustServiceRateAction, which is used to adjust the service rate of an invoked SLS depending on the actual conditions of the network. This class is defined in a similar manner as the SLSMSubPolicyDowngradeServiceRateAction where the administrator can define the adjusted service rate either in an absolute value (kbps) or as a percentage of the contractual value. The SLSMInvPolicyAdjustQoSAction is again a class defined in a similar fashion to SLSMSubPolicyDowngradeQoSAction and represents an action similar to a remarking action, giving the opportunity to the administrator to either explicitly specify the new QoS class the traffic will experience or choose a predefined strategy that will determine this QoS class such as the one with the most available bandwidth at the moment of the invocation based on actual monitoring information. The last policy action class defined is the SLSMInvPolicyAdjustAdmissionAction, which is used to adjust the admission control parameters

with the aim of avoiding overwhelming the network. The values of the properties will determine the severity of the action as described in section 5.1.3.2 and are defined with regard to the values of R^{a}_{min}, R^{w}_{min} and R_{max} present in the RAM. The properties spAdmissionMinThreshold and spAdmissionMaxThreshold define the minimum and maximum threshold of the RED like admission control algorithm in % of the different ranges present in the RAM as specified by the spRangeInRAM property. Four ranges are defined in the RAM. The first one covers the areas from 0 to R^{a}_{min}, the second one covers the area from R^{a}_{min} to R^{w}_{min}, the third one is the area between R^{w}_{min} and R_{max} and last one is the remaining capacity above R_{max}.

5.1.4.2 SLS Management Policy Conditions

The SLS Management Policy Conditions are modelled using the abstract class PolicyCondition defined in PCIM, the SimplePolicyCondition class defined in PCIMe and classes that we explicitly defined, derived from the PolicyCondition class. The SLS management conditions capture the requirements of SLS subscriptions and invocation requests so that differential treatment can be applied based on the policy actions defined above. Besides the conditions that enable differential treatment of the subscriptions and invocations, conditions are also defined that represent the level of the subscribed and invoked traffic demand with respect to the RAM as well as the network status in order to guide the behaviour of subscription/invocation admission control. The class inheritance hierarchy of the SLS Subscription and Invocation Management policy conditions is shown below.

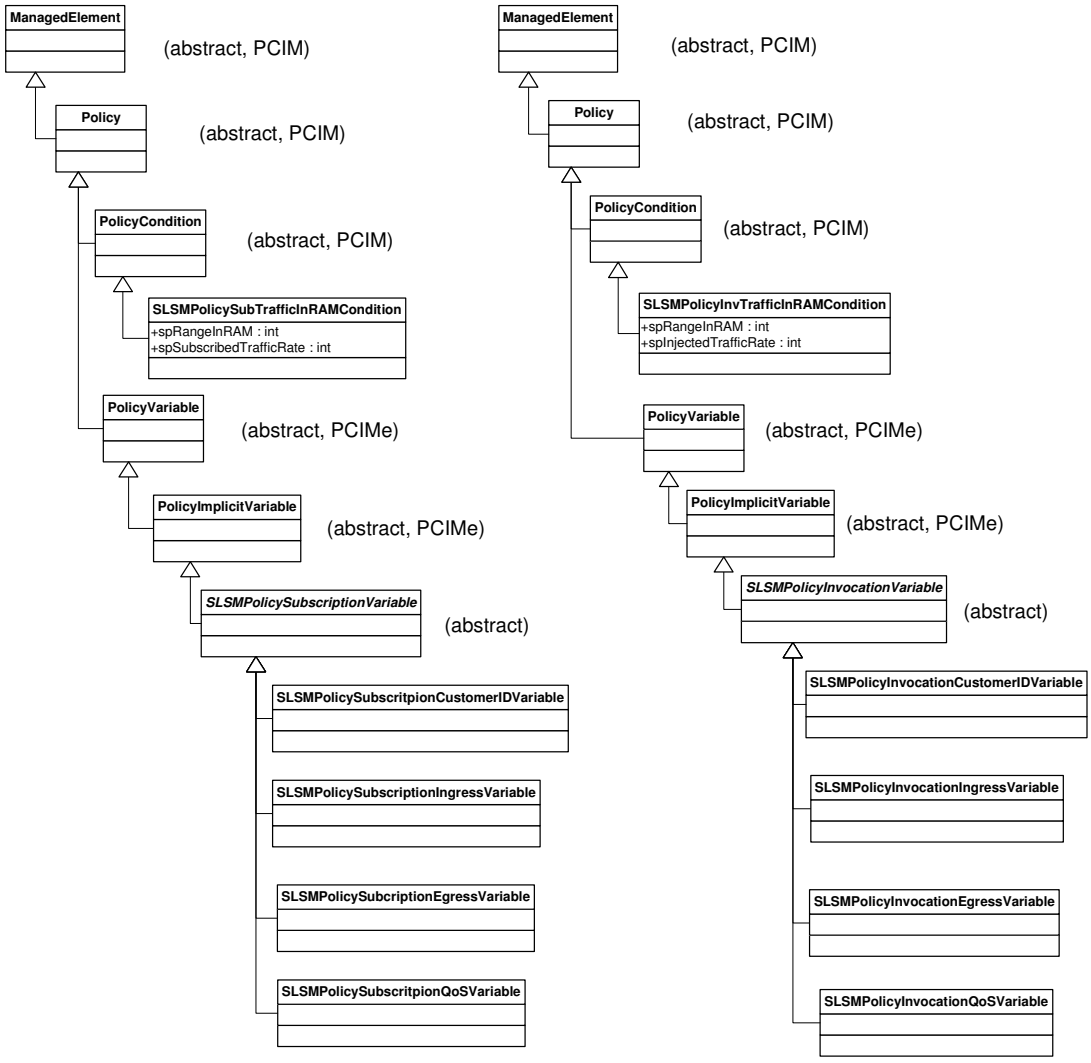


Figure 5-4 SLS Management Policy Conditions – Class Inheritance Hierarchy

For the conditions that are modelled using the SimplePolicyCondition class, we defined variable classes, reusing the mechanism defined in PCIMe, which represent parameters of the SLS subscription and Invocation request. SLS subscription and SLS invocation related variable classes extend the SLSMPolicySubscriptionVariable and SLSMPolicyInvocationVariable respectively which they extend the PolicyImplicitVariable class, defined in PCIMe. Subclasses specify the data type and semantics of the policy variables. For the purpose of this work, we focused only on variables that we envisage to be mostly used in SLS management policy conditions. These include the customer ID that initiated the subscription and invocation request modelled by SLSMPolicySubscriptionCustomerIDVariable and SLSMPolicyInvocationCustomerIDVariable respectively. The scope of the SLS which is either requested to be subscribed or invoked is modelled by two variables. The first one represents the ingress node of the domain that the customer is attached to and the other represents the egress node where the traffic to be injected by

the customer will exit the operator's domain. These are modelled by the SLSMPolicySubscriptionIngressVariable, SLSMPolicySubscriptionEgressVariable for the subscription requests and by the SLSMPolicyInvocationIngressVariable and SLSMPolicyInvocationEgressVariable for the invocation requests. Finally, the QoS Class to be offered to traffic originated by the customer that initiated the subscription and invocation request is modelled by the SLSMPolicySubscriptionQoSVariable and SLSMPolicyInvocationQoSVariable respectively. The network status is also another condition modelled by a variable defined by the SLSMPolicyInvNetworkCongestionVariable class to enable invocation admission control decisions based on the alarms received by the network monitoring component. When the latter variable's value is true then the network status is considered to be congested (red) while when it is false it is considered be green as defined in section 5.1.3.2.

Since admission control decisions both at the subscription and invocation level are based on the level of forecasted/injected traffic projected on the RAM provided by the offline TE component, we defined two classes to represent these policy conditions that evaluate to true when this defined level is crossed: the SLSMPolicySubTrafficInRAMCondition class and the SLSMPolicyInvTrafficInRAMCondtion class. These classes are derived from the abstract class PolicyCondition defined in PCIM and contain two properties in order to enable the definition of the level of the subscribed/invoked traffic with respect to the different ranges in RAM. The property spRangeInRAM defines the range in the RAM against which the value of the subscribed/injected traffic rate is defined. Four ranges are defined in the RAM. The first one covers the areas from 0 to R^{a}_{min}, the second one covers the area from R^{a}_{min} to R^{w}_{min}, the third one is the area between R^{w}_{min} and R_{max} and last one is the remaining capacity above R_{max}. Then the property spSubscribedTrafficRate/spInjectedTrafficRate is used to define the level in % of the ranges in the RAM as defined by the value of the property spRangeInRAM.

5.2 Traffic Engineering Policies

5.2.1 System Architecture

In Figure 5-5, we depict only the Traffic Engineering part of the architecture presented in Chapter 4 together with the components of the policy management sub-system in the same manner as presented for the SLS management part. A single policy consumer is needed for driving the behaviour of Network Dimensioning while policy consumers are distributed in every edge router for managing the DRtM component and policy consumers driving the behaviour of DRsM components are distributed in every node of the managed network. Based on the classification of

the policies driving the behaviour of the QoS management architecture (section 4.8) we show the interactions of the policy consumers with the monitoring components required for receiving policy triggering events which depend on the network state.

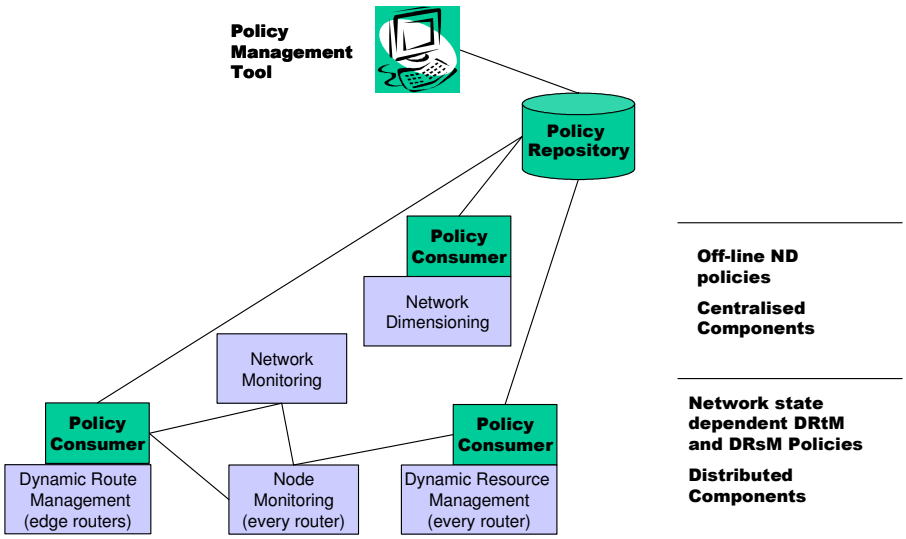


Figure 5-5 Policy-driven Traffic Engineering System Architecture

Due to the fact that ND is an offline component, the corresponding policy consumer does not need to communicate with any other external component since policies are triggered based on internal events e.g. the arrival of a new Traffic Matrix from the Traffic Forecast component. On the other hand, the policy consumer that drives the behaviour of the DRtM component needs to communicate both with the network monitoring in order to receive events regarding the status of edge-to-edge LSPs, e.g. if their QoS deteriorates, and the node monitoring for getting information on the usage of the currently invoked services. In the same manner, the policy consumer attached to the DRsM component interacts with the node monitoring component for receiving threshold crossing events for the bandwidth usage of the PHBs. The specific functionality and algorithms presented here are based on the work published in [Trim03] and [D1.2]. The Traffic Engineering policies are further explained in the sections below.

5.2.2 Network Dimensioning

5.2.2.1 Functionality Description

ND performs the provisioning activities of the system. It is responsible for the long to medium term configuration of the network resources. By configuration we mean the definition of LSPs as well as the anticipated loading for each PHB on all interfaces, which are subsequently being translated by DRsM into the appropriate scheduling parameters (e.g. priority, weight, rate limits) of the underlying PHB implementation. ND does not provide absolute values but they are in the

form of ranges, constituting directives for the function of the PHBs, while for LSPs they are in the form of multiple paths to enable multi-path load balancing. The exact PHB configuration values and the load distribution on the multiple paths are determined by DRsM and DRtM respectively, based on the state of the network, but should adhere to the ND produced directives.

ND runs periodically, first requesting the predictions for the expected traffic per QoS Class in order to be able to compute the provisioning directives. The dimensioning period is in the time scale of a week while the forecasting period is in the time scale of hours. The latter is a period in which we have considerably different predictions as a result of the time schedule of the subscribed Service Level Specifications (SLSs). For example, ND might run every Sunday evening and provide multiple configurations i.e. one for each period of each day of the week (morning, evening, night).

The objectives are both traffic and resource-oriented. The former relate to the obligation towards customers, through the SLSs. These obligations induce a number of restrictions about the treatment of traffic. The resource-oriented objectives are related to the network operation, more specifically they are results of the high-level business policy that dictates the network should be used in an optimal manner. The basic Network Dimensioning functionality is summarized in the following table.

Table 5-3 Network Dimensioning Algorithm Overview

Input:
Network topology, link properties (capacity, propagation delay, PHBs)
Pre-processing:
Request traffic forecast, i.e. the potential traffic trunks (TT)
Obtain statistics for the performance of each PHB at each link
Determine the maximum allowable hop count κ per TT according to the PHB statistics
Optimisation phase:
Start with an initial allocation (e.g. using the shortest path for each TT)
Iteratively improve the solution such that for each TT find a set of paths:
The minimum bandwidth requirements of the TT are met
The hop-count constraints κ is met (delay/ loss requirements are met)
The overall cost function is minimized
Post-processing:
Allocate any extra capacity to the resulted paths of each OA according to resource allocation policies
Sum the path requirements per link per OA, give minimum (optimisation phase) and maximum (post-processing phase) allocation directives to DRsM
Give the appropriate paths calculated in the optimisation phase to DRtM
Store the configuration into the Network Repository

The network is modelled as a directed graph $G = (V, E)$, where V is a set of nodes and E a set of links. With each link $l \in E$ we associate the following parameters: the link physical capacity

C_l, the link propagation delay d_l^{prop}, the set of the physical queues K, i.e. Ordered Aggregates (OAs), supported by the link. For each OA, $k \in K$ we associate a bound d_l^k (deterministic or probabilistic depending on the OA) on the maximum delay incurred by traffic entering link l and belonging to the $k \in K$, and a loss probability p_l^k of the same traffic.

The basic traffic model of ND is the traffic trunk (TT). A traffic trunk is an aggregation of a set of traffic flows characterized by similar edge-to-edge performance requirements i.e. belonging to the same QoS Class. Also, each traffic trunk is associated with one ingress node and one egress node, and is unidirectional. The set of all traffic trunks is denoted by T .

The *primary objective* of such an allocation is to ensure that the requirements of each traffic trunk are met as long as the traffic carried by each trunk is at its specified minimum bandwidth. However, with the possible exception of heavily loaded conditions, there will generally be multiple feasible solutions. The design objectives are further refined to incorporate other requirements such as:

a) avoid overloading parts of the network while other parts are under loaded,

b) provide overall low network load (cost).

The last two requirements do not lead to the same optimisation objective. In any case, in order to make the last two requirements more concrete, the notion of "load" has to be quantified. In general, the load (or cost) on a given link is an increasing function of the amount of traffic the link carries. This function may refer to link utilization or may express an average delay, or loss probability on the link. Let x_l^k denote the capacity demand for OA $k \in K$ satisfied by link l and $u_l^k = x_l^k / C_l$ the link utilisation. Then the link cost induced by the load on OA $k \in K$ is a convex function, $f_l^k(u_l^k)$, increasing in u_l^k. The total cost per link is defined as $F_l\left(\overline{u_l}\right) = \sum_{k \in K} f_l^k\left(u_l^k\right)$, where $\overline{u_l} = \left\{u_l^k\right\}_{k \in K}$ is the vector of demands for all OAs of link l. The total cost per link is an approximate function, e.g. $f_l^k(u_l^k) = a_l^k u_l^k$.

We provide an objective that compromises between the two a) and b), that is avoid overloading parts of the network and minimize overall network cost:

$$\text{minimise} \sum_{l \in E}\left(F_l\left(\overline{u_l}\right)\right)^n = \sum_{l \in E}\left(\sum_{k \in K} f_l^k\left(u_l^k\right)\right)^n, n \geq 1 \qquad \text{(Eq.1)}$$

When $n = 1$, the objective (Eq.1) reduces to objective b), while when $n \to \infty$ it reduces to a).

Each traffic trunk is associated with an end-to-end delay and loss probability constraint of the traffic belonging to the trunk. Hence, the trunk routes must be designed so that these two

constraints are satisfied. Both the constraints above are constraints on additive path costs under specific link costs. However the problem of finding routes satisfying these constraints is, in general, NP-complete [Wang96]. Given that this is only part of the problem we need to address, the problem in its generality is rather complex.

Usually, loss probabilities and delay for the same PHB on different nodes are of similar order. We simplify the optimisation problem by transforming the delay and loss requirements into constraints for the maximum hop count for each traffic trunk (TT). This transformation is possible by keeping statistics for the delay and loss rate of the PHBs per link, and by using the maximum, average or n -th quantile in order to derive the maximum hop count constraint.

For each traffic trunk $t \in T$ we denote as R_t the set of (explicit) routes defined to serve this trunk. For each $r_t \in R_t$ we denote as b_{r_t} the capacity we have assigned to this explicit route. We seek to minimize (Eq. 1), such that the hop-count constraints are met and the bandwidth of the explicit routes per traffic trunk should be equal to the trunks' capacity requirements.

This is a network flow problem and considering the non-linear formulation, for the solution we use a customised version [Trim03b] of the gradient projection method [Berts99]. This is an iterative method, where we start from an initial feasible solution, and at each step we find the minimum first derivative of the cost function path and we shift part of the flow from the other paths to the new path, so that we improve our objective function (Eq. 1). If the path flow becomes negative, the path flow simply becomes zero. This method is based on the classic unconstraint non-linear optimisation theory, and the general point is that we try to decrease the cost function through incremental changes in the path flows. A more detailed description of the algorithm is presented in [Trim03b].

5.2.2.2 Network Dimensioning Policies

ND is triggered periodically and does not depend on network state. As such, policies that are enforced on this component are not triggered from events that occur within the network. We identify two categories of policies for such a static, off-line resource management component. The first category, *initialisation policies*, concerns policies that result in providing initial values to variables, which are essential for the functionality of ND and do not depend on any state but just reflect decisions of the policy administrator. The second category, *resource provisioning policies*, concerns those that influence the way it calculates the capacity allocation and the path creation configuration of the network. Such policies are those that their execution is based on the input from the traffic forecast module and on the resulting configuration of the network. We discuss the directives/constraints specified as policies that should be taken into account when the dimensioning component is calculating a new configuration following the categorisation

discussed above. A summary of Network Dimensioning policies defined in this section is shown in the Table below, describing the policy actions and the parameters based on which policy conditions are formed.

Table 5-4 Summary of Network Dimensioning Policies

Network Dimensioning Policies	
Policy Actions	**Policy Conditions**
Initialisation Actions (setting variables)	**Traffic Matrix/Traffic Trunk parameters**
- Dimensioning Period	- QoS Class
- Cost Function	- Ingress, Egress(es)
- Optimisation objective	- Bandwidth
Resource Provisioning Actions	**Post Calculation parameters**
- Bandwidth Allocation (min, max , range)	- Spare Bandwidth
- Post Processing Bandwidth Treatment	- Excess Bandwidth
- Hop Count Derivation	- Maximum Link Load
- Explicit Route Setup	- Network Load
- Maximum Alternative Trees Definition	

Since dimensioning is triggered mainly periodically, the policy administrator should specify this period, which specifies when ND will get the traffic forecast. The priority of this policy should be specified in order not to cause any inconsistencies when re-dimensioning is triggered by notifications from the dynamic control parts of the system, that is when DRtM and DRsM are unable to perform an adaptation of the network with the current configuration. Another parameter that should be defined by policies is the cost-function used by ND. The administrator should be able either to choose between a number of pre-specified cost functions and/or setting values to parameters in the desired function. For example, if the approximate cost function used by ND is linear to the bandwidth allocated to a PHB, that is $f_l^k(x_l^k) = a_l^k x_l^k$ where x_l^k is the bandwidth allocated to PHB k on link l and a_l^k is a constant, the value of this constant could be specified by the policy administrator depending on the cost, i.e. importance, of a particular PHB. A more flexible approach would be to allow the administrator to specify with the policy a different cost function to be used. Another constraint that policies may add to ND is that of the maximum allowed number of hops of TTs. The administrator should have the flexibility to specify the

maximum number of hops that routes are permitted to have. This number may vary depending on the QoS class the TT belongs to. Another constraint imposed by policies is the maximum number of alternative paths that ND defines for every traffic trunk for the purpose of load balancing. Finally, the exponent n, as defined in Equation 1, is another parameter that is specified by policies allowing the administrator to choose the relative merit of low overall cost and avoid overloading parts of the network.

Example 5.5 `If maxLinkLoad > 80% then dimension with minimising MaxLinkLoad objective`

The above policy rule causes network dimensioning to calculate a new configuration with a different value of the cost function exponent if the maximum link load of the produced configuration is above 80%.

The policy administrator should be able to specify the amount of network resources (giving a minimum, maximum or a range) that should be allocated to each traffic type i.e. Expedited Forwarding, Assured Forwarding, and Best Effort. This will cause ND to take into account this policy when calculating the new configuration for this QC together with the information produced by the traffic forecast. More specifically, ND should allocate resources in a way that does not violate the policy and then calculate the configuration taking into account the remaining resources. A more flexible option would be for the policy administrator to indicate how the resources should be shared in specific (critical) links. After the optimisation phase ends (see Table 5-3), ND enters a post-processing stage where it will try to assign the residual physical capacity to the various QCs. This distribution of spare capacity is left to be defined by policies that indicate whether it should be done proportionally to the way resources are already allocated or it can be explicitly defined for every traffic class. A similar policy to the previous one would be to specify the way the capacity allocated to each QC should be reduced because the link capacity is not enough to satisfy the predicted traffic requirements. Since ND actually translates the delay and loss requirements on an upper bound on the number of hops per route, the way this translation is done can also be influenced by policy rules. For example, the safest approach to satisfy the TT requirements would be to assume that every link and node belonging to the route induces a delay equal to the maximum delay caused by a link and node along the route. So, this policy rule will allow the administrator to decide if the maximum, average or minimum delay or loss induced by a node or link along the route should be used to derive the hop count constraint according to the QC that the TT belongs to. Policies that allow the administrator for a particular reason to explicitly specify an LSP that a TT should include can also be defined. Of course, this should override the algorithm's decision about the creation of the LSP for this TT and it should continue to run for the rest of the entries in the traffic matrix.

Example 5.6 `if TT_QC==EF and TT_Ingress==4 and TT_Egress==6 then Setup LSP 4-9-7-6 2000 kbps`

The above policy rule creates an explicit LSP following the nodes 4, 9, 7, 6 with the bandwidth of the TT being 2 Mbps that is associated with this LSP in the case ND receives a Traffic Matrix that contains a TT that enters the network from ingress node 4 and exits the domain from node 6.

5.2.3 Dynamic Resource Management

5.2.3.1 Functionality Description

There are two main resources that need to be managed for the router to handle traffic at the packet level: link bandwidth and buffer space. While controls are exercised in order to ensure the proper allocation of these two resources, one should keep in mind a third resource that enables the control process, namely processing power. The limited amount of the latter resource, dictates to avoid overly complicated control processes whose on-line operation would require inordinate amount of time, hence defeating the purpose of timely resource allocation. In the work presented here, we concentrate only on how DRsM manages the distribution of link bandwidth to the different supported PHBs.

DRsM is responsible for tracking the utilisation of each PHB and ensuring that the bandwidth allocated to the PHB is in accordance with the bandwidth required. DRsM determines the required bandwidth for each PHB according to observed utilisation. When any PHB is under-utilised DRsM should reduce its bandwidth share so that the saved bandwidth can be reallocated to other PHBs that require more bandwidth than they have been allocated. If a PHB is overloaded then the bandwidth should be increased accordingly if sufficient link capacity is available.

ND defines upper and lower bounds for the bandwidth to be allocated to each PHB. If DRsM sees that the PHB should be allocated bandwidth outside this range, it will raise an alarm to ND. ND is then responsible for reconfiguring the upper and lower bounds if necessary.

Monitoring PHB utilisations is achieved through the node monitoring component. Rather than polling instantaneous values, DRsM defines upper and lower thresholds of the bandwidth consumed by a PHB. Node monitoring will raise a threshold crossing alarm when the bandwidth exceeds the upper threshold or drops below the lower threshold. On receiving these threshold crossing alarms, DRsM will increase or reduce the bandwidth allocated to that PHB and re-determine the monitoring thresholds accordingly.

5.2.3.2 Dynamic Resource Management Policies

We present below an analysis of the policies that an administrator could introduce in order to drive the behaviour of the DRsM component described above. Figure 5-6 depicts the functionality that can be achieved through the execution of such policies.

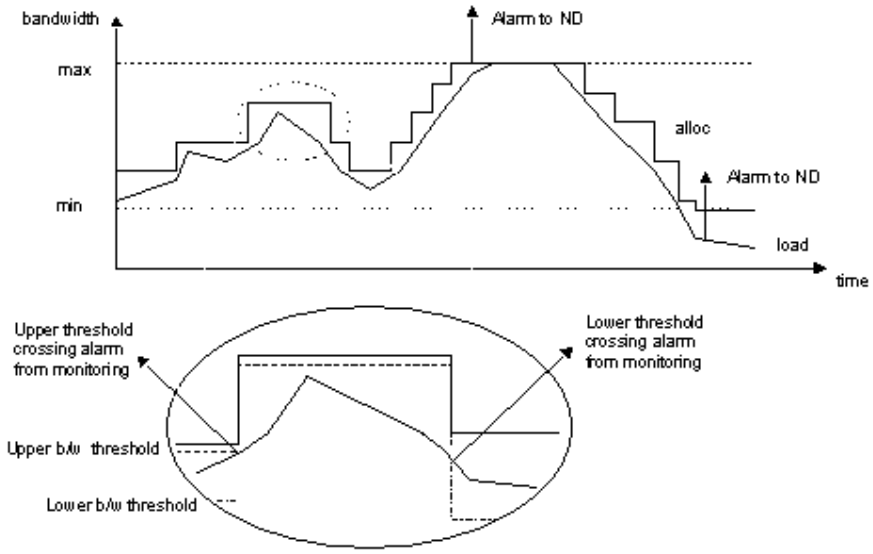


Figure 5-6 Bandwidth tracking of a single PHB

A summary of the Dynamic Resource Management policies defined in this section is shown in Table 5-5, describing the policy actions and the parameters based on which policy conditions are formed.

The first category of Dynamic Resource Management functionality guided by policies is the definition of the thresholds to the monitoring component. The calculation of the upper and lower bandwidth thresholds, $uthresh_p$ and $lthresh_p$, should be defined above and below the current load ($load_p$). The distance from $load_p$ can be based on fixed absolute or relative values defined by a policy. Alternatively, it can be predicted through extrapolation based on the historical traffic load. Moreover, policies can be specified in order to dynamically calculate the thresholds, depending on how accurately the bandwidth needs to be tracked or on how frequently DRsM wishes to receive threshold crossing alarms, e.g. by introducing a type of time window such that at most once warning will be taken into account within each time period.

Table 5-5 Summary of Dynamic Resource Management Policies

Dynamic Resource Management Policies	
Policy Actions	**Policy Conditions**
Threshold Setting - Absolute value(kbps), Relative values (%) - Extrapolation **Required Bandwidth Calculation** - Increase/Decrease by an absolute value - Extrapolation **Post Calculation** - Spare Bandwidth Treatment - Excess Bandwdith Treatment	**Threshold Crossing parameters** - PHB - Upper Threshold - Lower Threshold **Post Calculation parameters** - Spare Bandwidth - Excess Bandwidth

The policy administrator should be able to specify policies that influence the way the required bandwidth is calculated. These policies will be triggered when a threshold is crossed and compute the required bandwidth either by increasing the bandwidth by an absolute or relative value (e.g. 5% and 10 %) in case of upper threshold crossings, or decreasing the bandwidth in case of lower threshold crossings. Another approach would be to calculate a predicted value for the bandwidth required in the next time interval according to trend analysis of historical data regarding the particular PHB utilization. A well known method would be to use an Exponentially Weighted Moving Average (EWMA) approach providing even more flexibility by setting parameters such as the size of the extrapolation window, the number of historical data to be used in the extrapolation function, etc. Another aspect that could be left to the administrator to define through policies is whether to calculate the required bandwidth of the particular PHB that the threshold crossing occurred or calculate the required bandwidth of all PHBs every time a threshold is crossed.

Example 5.7 `if PHB == EF and Upper_Threshold_Crossed == True then Increase Required_Bw by 10%`

The above policy rule states that when the upper threshold for the EF PHB is crossed then the required bandwidth should be calculated by increasing the bandwidth by 10%.

While the above policies related to bandwidth tracking illustrate the role of managing a single PHB, the complete objective of Dynamic Resource Management policies would be to manage the resources for all PHBs on a link, dictating the distribution of the link's resources among them according to business objectives. After calculating the required bandwidth based on the policies described above, policies can be introduced defining the way spare capacity should be distributed if the sum of the bandwidth demands is less than the link capacity. In the same manner, policies related to the distribution of bandwidth when the sum of required bandwidth exceeds the link capacity could also be defined. For example, a policy could be to allocate a minimum bandwidth share to each PHB (defined by ND) when congestion occurs and distribute the remaining bandwidth either equally or proportionally to the PHBs that require more bandwidth than that has been allocated. Finally, policies can be introduced that specify the conditions that should be met in order to issue alarms to ND, such as when the required bandwidth for a PHB is bigger than the maximum value provided by ND.

Example 5.8 `if LinkSpareBw == True then Equally_Distribute_Bw`

The above policy rule states that after the required bandwidth is calculated and there is spare bandwidth on the link, then this should be equally distributed to the supported PHBs.

5.2.4 Dynamic Route Management

5.2.4.1 Functionality Description

In the MPLS approach, the Dynamic Route Management (DRtM) component is a distributed component located at the edge routers, responsible for managing the routing processes in the network according to the guidelines provided by Network Dimensioning. This amounts to:

- Setting up traffic forwarding parameters at the ingress node, so that incoming traffic is routed to LSPs according to the bandwidth determined by Network Dimensioning.
- Modifying the routing of traffic according to feedback received from Network Monitoring
- Issuing alarms/warnings to Network Dimensioning in case available capacity cannot be found to accommodate new connection requests

During initialisation, Network Dimensioning provides DRtM the set of traffic trunks, which are to be managed with DRtM. The common characteristic of this set of traffic trunks is that they all have as ingress node the node for which the given DRtM is responsible. With each traffic trunk (TT) the following information is provided:

- The set of LSPs to which traffic belonging to the TT is to be routed, as well as the bandwidth of each of these LSPs (the bandwidth of an LSP is logically allocated to it only at he ingress node i.e. it is not hard-assigned to all the links across the path).
- The PHB treatment of traffic that belongs to the TT.
- The end-to-end delay and loss probability (upper bound) of traffic that belongs to the TT.

One of the main requirements imposed on the routing of traffic that belongs to a traffic trunk is that packets that belong to an application should not arrive out of order at the egress router under normal conditions. This requirement imposes a constraint on the manner in which routing is to be done. In order to avoid out of order packet arrivals, the initial approach is to perform routing based on (a prefix of) flow identification (e.g. origin id, destination id). Hence with each LSP, a group of flow ids are associated and traffic from these flows is routed on the LSP.

The manner in which the mapping of the flows to LSPs is made may have a significant effect on the overall performance of the network. Also a proper initial mapping may avoid frequent traffic rerouting and forwarding table updates during system operation. Since at system initialisation the available statistical information for each traffic trunk is the bandwidth of the address groups, the mapping is based on associating traffic from groups to a given LSP.

The above problem is NP-complete since a special case of it is the bin-backing problem. Hence solutions must resort to heuristic algorithms. An algorithm proposed in [D1.2] is based on the following observations. It is easier to find a good allocation if the address group bandwidths are small (relative to LSP bandwidths). Hence one should try to accommodate address groups with large bandwidth fist so that there is more flexibility as to where to place these address groups. Allocating address groups first to LSPs with large bandwidths results in larger remaining unallocated LSP bandwidth and hence allows for more flexibility for subsequent SLS allocations.

The mapping of flows to LSPs is semi-permanent in the sense that DRtM may decide to alter this mapping based on network conditions. Since we follow a policy driven approach for guiding the behaviour of the components, the above decision is guided by policies introduced by the administrator which are described in the next section.

5.2.4.2 Dynamic Route Management Policies

Policies managing the DRtM components located at the edge routers of the managed network are based on network events and conditions related to the performance of the established LSPs initiated by the edge router of the associated DRtM. In order to perform proper load balancing, policies that perform rerouting actions should be enforced, triggered based on monitoring two types of QoS metrics: PHB QoS performance and end-to-end LSP QoS performance. A summary

of the Dynamic Route Management policies defined here is shown below, describing the policy actions and the parameters based on which policy conditions are formed.

Table 5-6 Summary of Dynamic Route Management Policies

Dynamic Route Management Policies	
Policy Actions	**Policy Conditions**
Rerouting Actions - Explicitly route addresses to paths **Determine alternative paths** - Based on QoS characteristics (delay, loss) - Based on Available Bandwidth	**Node PHB QoS** - Delay, Packet Loss Rate **End-to-End LSP QoS** - Delay, Packet Loss Rate **LSP Available Bandwidth**

Enforcement of policy actions triggered by monitoring events of PHB QoS performance, aim to take proactive measures. Specifically, their target is to avoid routing traffic to LSPs using the PHBs whose QoS performance in terms of delay and loss probability becomes critical, even though end-to-end performance deterioration on these LSPs may not have been observed. Hence, actions at this stage attempt to avoid the deterioration of end-to-end QoS metrics and in addition help in relieving the load on the congested PHB. For example, when the measured PHB delay on a node exceeds an upper threshold, then actions that reroute a set of address prefixes that are mapped to one or a set of LSPs using the critical PHB could be enforced. The address prefixes can either be explicitly defined or a set can be determined that their collective bandwidth exceeds a specific threshold. The load incurred by various address prefixes is provided by Network Monitoring statistics, which are based both on existing SLS contracts and on real measurements. The rerouting policy action determines an accessible LSP i.e. LSP that does not pass through PHBs whose performance starts deteriorating among those that belong to the same multipath as defined by ND. Different strategies can be applied for this decision e.g. the LSP with the most available bandwidth to be selected until the latter drops below an administrator set threshold.

Policies that are triggered based on events about end-to-end LSP QoS performance aim to take reactive measures. We need to distinguish two cases. The first case concerns LSP bandwidth depletion while the second concerns end-to-end QoS metrics. For the first case, rerouting policy actions will be enforced when an alarm is received regarding a LSP that its bandwidth is being depleted i.e. has crossed a lower threshold. The amount of address prefixes may be chosen in order for the used bandwidth of that LSP to reach a specific threshold. For the second case, rerouting actions will be enforced when network monitoring issues an alarm denoting that an LSP

is in critical condition regarding its QoS i.e. the delay or loss rate crosses a specific threshold. The rerouting strategies enforced for the two cases described previously could be the same as the ones presented for the case of PHB QoS performance deterioration. Of course the alternative LSPs that are selected are based on their available bandwidth and the level of QoS respectively.

Example 5.9 `if PHB == EF and LSP_Available_Bw < 10% then reroute_addresses_Bw_amount 15% to most_available_bw LSPs`

The above policy rule states that if the bandwidth of LSPs that are used by EF traffic have less than 10% available bandwidth then reroute addresses that consume 15% of the bandwidth should to alternative LSPs with the most available bandwidth.

Based on the information received by Network Monitoring, the policies described above may reassign some of the address groups to the various managed paths and hence update the LSP forwarding table at the ingress router. During this process, mechanisms are employed to ensure that during reassignment the packets-in-order condition is satisfied. Finally, policies can be applied so that if appropriate LSPs for the reassignment cannot be found, alarms are issued to Network Dimensioning, which in turn may take more global actions in order to relieve the experienced congestion.

5.2.5 PCIM extensions for Traffic Engineering Policies

This section proposes an object-oriented information model for representing the Traffic Engineering policies presented in the previous sections based on the IETF Policy Core Information Model (PCIM) and its extensions (PCIMe). The Traffic Engineering Policy Information Model class inheritance hierarchy is rooted in PCIM and PCIMe.

5.2.5.1 Traffic Engineering Policy Actions

5.2.5.1.1 Network Dimensioning Policy Actions

Figure 5-7 depicts a part of the inheritance hierarchy of our information model representing the ND policy actions, while it also indicates its relationships to PCIM and PCIMe. Note that some of the actions are not directly modelled. Instead they are modelled by using the class SimplePolicyAction using instances of the variable and value classes (“SET <variable> TO <value>”).

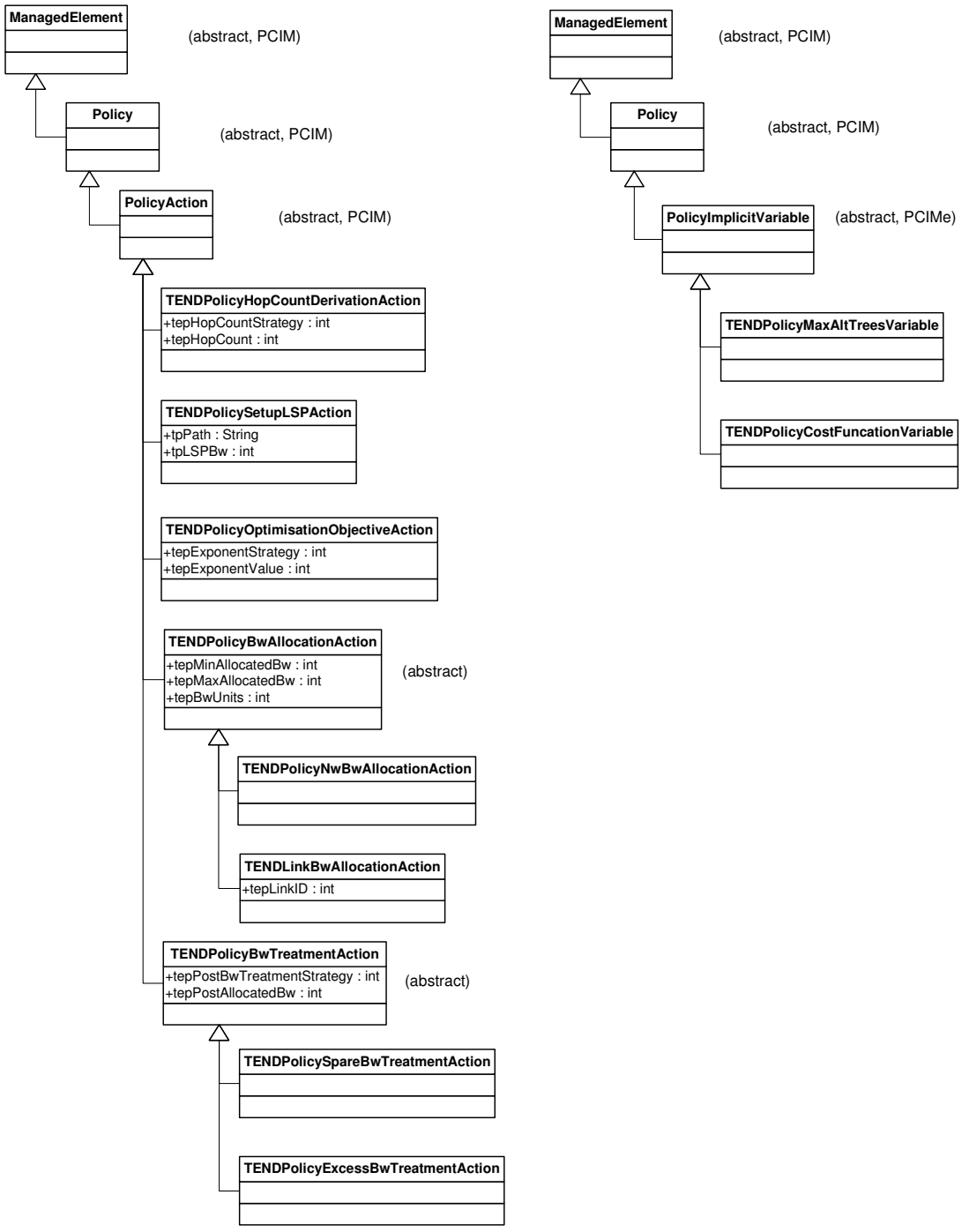


Figure 5-7 Network Dimensioning Policy Actions – Class Inheritance Hierarchy

The class TENDHopCountDerivationAction represents the policy action that will determine the strategy to be used at the pre-processing stage when ND translates the QoS requirements to maximum number of hop counts. The tepHopCountStrategy property of the aforementioned class is an enumerated integer and is used to define the strategy to be followed such as the minimum, average or maximum delay or loss induced by a node as well as the definition of an absolute value

defined by the tepHopCount property of this class. The policy action that enables the administrator to explicitly setup an LSP is represented by the TENDSetupLSPAction class. It contains two properties, the tepPath property that defines the nodes that the LSP should follow and tepLSPBw that represents the bandwidth in kbps that is logically allocated to this LSP. The policy action that defines the optimisation objective of ND when it calculates a new configuration is represented by the class TENDPolicyOptimisationObjectiveAction. The property tepExponentStrategy defines the strategy to be used for setting the value of the exponent as defined in the equation (1) where the value of 0 means that an explicit value will be provided by the property tepExponentValue. Other strategies include increasing or decreasing the value of the exponent favouring the two different objectives as defined previously in section 5.2.2.1. The action that defines the maximum alternative paths/trees to be calculated is represented using the SimplePolicyAction by a pair of TENDPolicyMaxAltTreesVariable and IntegerValue classes. Similarly the action that sets the constant parameter of the cost function is modelled by the TENDPolicyCostFunctionVariable class.

The policy actions that provide guidelines for the allocation of resources are represented by classes that are all derived from the abstract class TENDPolicyBwAllocationAction. The latter contains properties (tepMinAllocatedBw, tepMaxAllocatedBw, tepBwUnits) that enable the specification of minimum and maximum values both in absolute values (kbps) and in percentages of the network/link capacity. The derived classes TENDPolicyNwBwAllocationAction and TENDPolicyLinkBwAllocationAction represent the bandwidth allocation action enforced network wide and on specific links respectively. The latter class introduces a new property tepLinkID representing the specific link where the minimum/maximum allocation should be applied. In a similar manner, the post processing policy actions are modelled by classes that are derived by the abstract class TENDPolicyBwTreatmentAction. This contains properties that define the strategy to be followed (tepPostBwTreatmentStrategy) where either a proportional, equal or an absolute value will be provided as defined by the property tepPostAllocatedBw of the spare/excess bandwidth distribution/reduction. The concrete classes derived from TENDPolicyBwTreatmentAction, TENDPolicySpareBwTreatmentAction and TENDPolicyExcessBwTreatmentAction represent the actions that distribute the spare bandwidth or reduce the excess bandwidth respectively.

5.2.5.1.2 Dynamic Resource and Route Management Policy Actions

Figure 5-8 depicts the inheritance hierarchy of the classes defined to represent policy actions that can be enforced on the Dynamic Resource and Route Management components. All the policy actions are modelled with classes derived by the abstract PolicyAction class defined in PCIM.

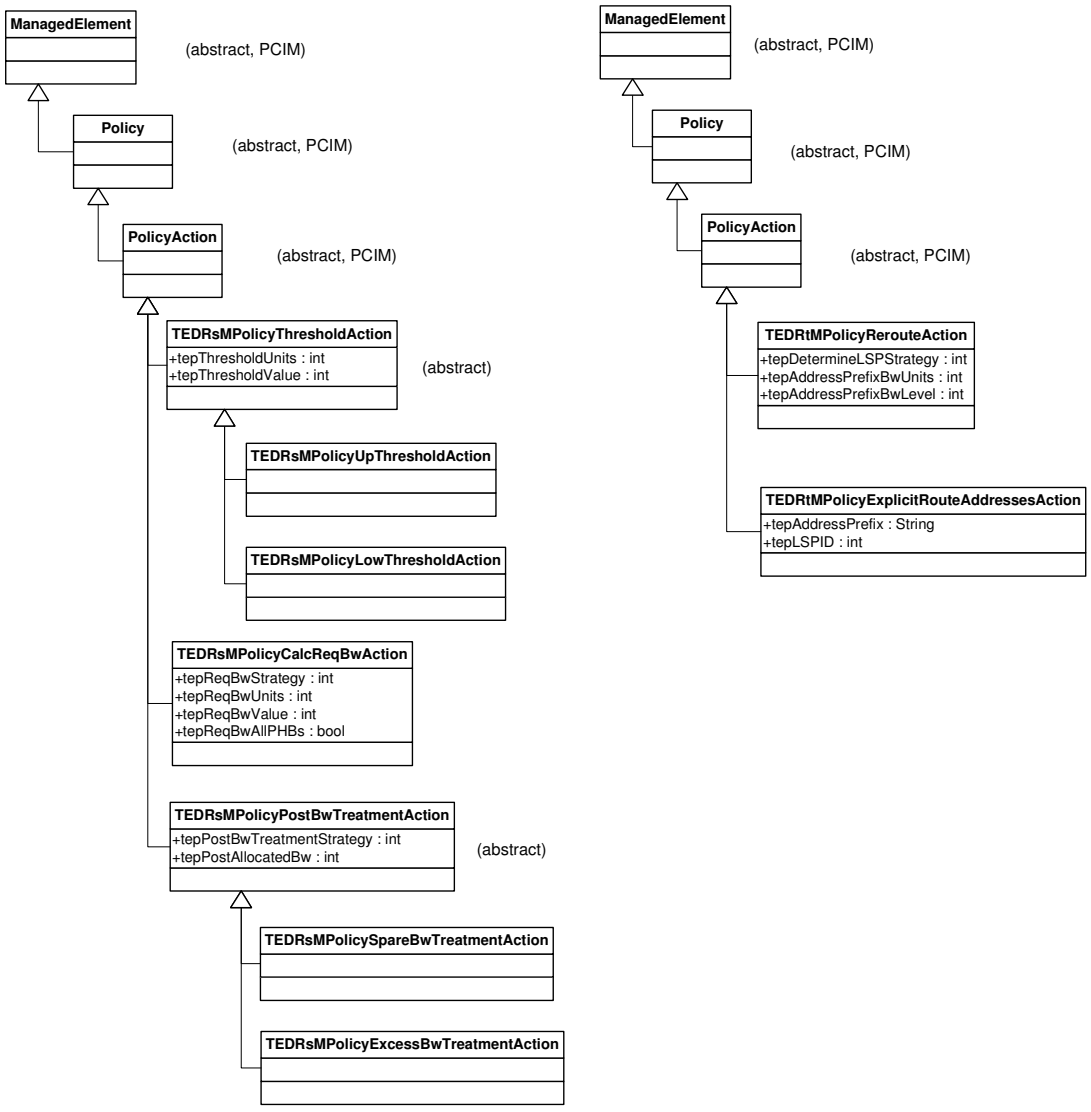


Figure 5-8 DRsM and DRtM Policy Actions – Class Inheritance Hierarchy

The DRsM policy actions that set the upper and lower thresholds to Node Monitoring are modelled by the classes TEDRsMPolicyUpThresholdAction and TEDRsMPolicyLowThresholdAction respectively. These are derived from the abstract class TEDRsMPolicyThresholdAction that contain properties that enable to set either an absolute (in kbps) or a relative value (%) with respect to the actual current load depending on the value of the tepThresholdUnits while the actual value is provided by the property tepThresholdValue. The policy actions that calculate the required bandwidth are modelled by the class TEDRsMPolicyCalcReqBwAction. The property tepReqBwStrategy included in this class defines the strategy to be followed for the calculation of the required bandwidth where the value of 0 means that this should be calculated by increasing the current allocation by an absolute value, 1 by decreasing by an absolute value and 2 by using an EWMA method. The actual absolute value

is either defined in % or in kbps by setting the appropriate values to the tepReqBwUnits and teqReqBwValue properties. The last property tepCalcReqBwAllPHBs defines whether to calculate the required bandwidth for all the PHBs or for only the PHB that a threshold has been crossed. The policy actions that define the way to distribute the spare bandwidth or reduce the excess bandwidth on every link to the supported PHBs are defined in the same fashion as the post bandwidth treatment actions applied to ND described in the previous section. These are modelled by the abstract class TEDRsMPolicyPostBwTreatmentAction and the derived classes TEDRsMPolicySpareBwTreatmentAction and TEDRsMPolicyExcessBwTreatmentAction.

There are two DRtM policy actions defined to represent the possible rerouting strategies to be enforced when certain conditions hold. The first one is the class TEDRtMPolicyRerouteAction. This class contains three properties: the tepDetermineLSPStartegy, the tepBwAddressUnits and tepAddressPrefixBwLevel. The tepDetermineLSPStartegy defines the strategy to be used in order to find an alternative path among those defined by ND. It is an enumerated integer where different values represent different strategies i.e. the LSP with the most available bandwidth, the smallest delay or loss. The tepAddressPrefixBwUnits specifies the units of the specified level of bandwidth consumed by a set of address prefixes defined by the property tepAddressPrefixBwLevel to be rerouted. Two values are possible either an absolute value in kbps can be specified or a percentage of the overall LSP (logically allocated) bandwidth. The second class TEDRtMPolicyExplicitRouteAddressesAction enables the explicit definition of the address prefixes (tepAddressPrefix) to be routed to a specific LSP (tepLSPID).

5.2.5.2 Traffic Engineering Policy Conditions

The Traffic Engineering Policy Conditions are modelled using the abstract class PolicyCondition defined in PCIM, the SimplePolicyCondition class defined in PCIMe and classes that we explicitly defined, derived from the PolicyCondition class as shown in Figure 5-9.

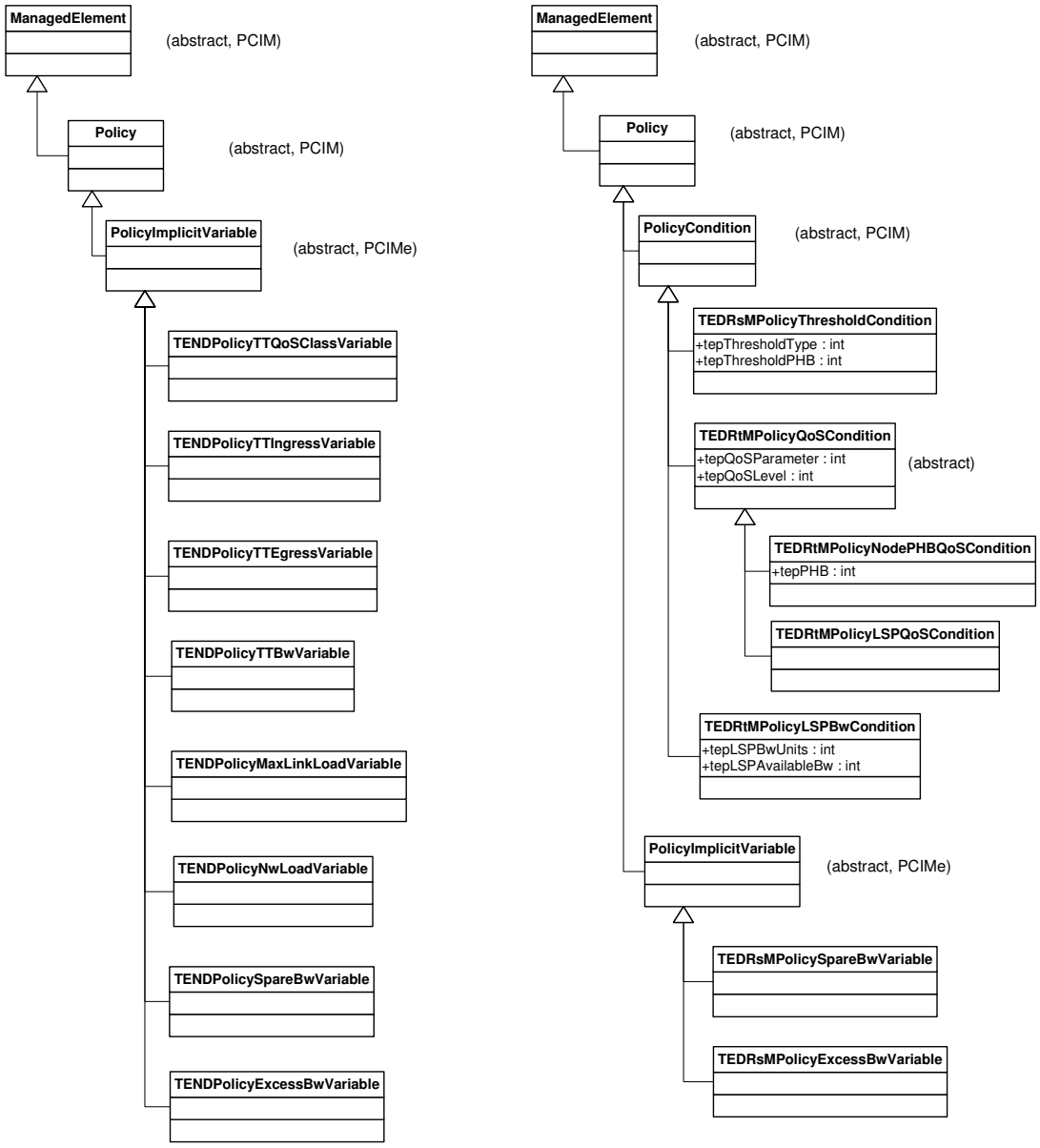


Figure 5-9 Traffic Engineering Policy Conditions – Class Inheritance Hierarchy

The policy conditions used in Network Dimensioning policies are mostly based on the characteristics of the traffic matrices and especially the traffic trunks which the ND component receives as input from the traffic forecast component. Depending on the attributes of the traffic trunks, different traffic engineering actions can be enforced on the ND component influencing its behaviour. These attributes are the QoS class, the ingress and the egress node as well as the associated bandwidth demand and are modelled by the variable classes TENDPolicyTTQoSClassVariable, TENDPolicyTTIngressVariable, TENDPolicyTTEgressVariable and TENDPolicyTTBwVariable class. Moreover, conditions based on the output configuration can be formed causing the optimisation algorithm to run until

the conditions are satisfied. Such conditions are modelled again by defining the variable classes TENDPolicyMaxLinkLoadVariable, TENDPolicyNwLoadVariable, TENDPolicySpareBwVariable and TENDPolicyExcessBwVariable. Time period conditions can also be applied influencing the behaviour of the ND component when it produces network configurations for different time periods.

The DRsM related policy conditions are based on the threshold crossing alarms produced by the node monitoring component and the values of the calculated required bandwidth. The TEDRsMPolicyThresholdCrossingCondition class is directly derived from the abstract PolicyCondition class and contains properties for describing the type of threshold crossing alarm. The property tepThresholdType is an enumerated integer, which can take two values representing either an upper or lower threshold crossing while the property tepThresholdPHB is again an enumerated integer describing for which PHB the threshold has been crossed. The variable classes TEDRsMSpareBwVariable and TEDRsMExcessBwVariable are variables that can be associated with Boolean values and are true when the sum of the required Bandwidths are less or greater than the link capacity respectively.

The DRtM Policy Conditions are related to the network conditions provided by the network and node monitoring component. These involve conditions modelled by classes which are derived by the PolicyCondition class. The abstract class TEDRtMPolicyQoSCondition is used as the basis for the conditions that are based on QoS performance characteristics. The property tepQoSparameter is an enumerated integer describing the different parameters to be monitored i.e. delay and packet loss rate while the property tepQoSLevel defines the level of the measured QoS performance that when crossed the condition will evaluate to true. The class TEDRtMPolicyNodePHBQoSCondition is a concrete class derived from the TEDRtMPolicyQoSCondition that represents the condition that evaluates to true when the QoS performance characteristics of a specific PHB in a node will exceed a specific level. The property tepPHB represents the PHB whose performance is monitored. The concrete class TEDRtMPolicyLSPQoSCondition is used to define conditions that evaluate to true when the end-to-end QoS performance of a LSP crosses a specific threshold. Finally, the class TEDRtMPolicyLSPBwCondition represents the condition that holds when the available bandwidth associated to this LSP drops below a specific level. The level of the available bandwidth can either be defined in kbps or in % of the overall bandwidth associated to this LSP depending on the values of the properties tepLSPBwUnits and tepLSPAvailableBw.

5.3 Hierarchical Policies case study: ND and DRsM policies

In this section, we provide a proof of concept validation of the methodology for applying policies to the hierarchical distributed system presented in Chapter 3 by presenting a case study of the hierarchical relationship between policies applied to the Network Dimensioning and Dynamic Resource Management components. We discuss in detail the case when a network dimensioning policy may be refined into a dynamic resource management policy reflecting, the two-level traffic engineering sub-system of our architecture.

Network Dimensioning policies influence the way the calculation of the network configuration is done at the beginning of every Resource Provisioning Cycle (RPC). Note that ND produces guidelines the dynamic resource management layer should follow within that RPC i.e. a minimum, a maximum and also a minimum share bandwidth in case of congestion for every PHB on every link of the managed network. The effect of an enforced ND policy is reflected in the guidelines produced by it. As such, DRsM policies may not be needed if the operation of the dynamic resource management layer is bound to the constraints produced by ND. When though the dynamic resource management functions operate in a less constrained fashion, possibly ignoring some of the ND guidelines, DRsM policies might be also enforced, complementing the management logic of the ND layer.

Let's assume that the administrator of an AS wants to ensure availability of a specific traffic class for a time period, for example s/he wants to enforce the following high-level policy:

"At least 20% of Network Resources should always be available for EF traffic"

By enforcing the corresponding policy at the ND level (shown below) the guidelines produced will reflect the operator's requirement i.e. the minimum value for EF distributed to every DRsM component at the routers would be at least 20 % of the link capacity.

Example 5.10 (ND Policy): **`if`** `TT_Qos_Class == EF` **`then`** `Network_Bw_Allocation > 20%`

On the other hand, if the dynamic resource management layer operates without taking into account this constraint, another policy should be enforced at the DRsM component (shown below), restricting it not to allocate less that 20 % of the link capacity to the EF PHB.

Example 5.11 (DRsM Policy): **`if`** `PHB == EF` **`and`** `Required_Bw < 20%` **`then`** `Required_Bw == 20%`

Of course the guidelines produced by ND are based on an optimisation algorithm that has a centralised network-wide view; hence if the dynamic functions utilise this information will probably lead the network to near-optimal levels in cases of congestion. On the other hand, these

constraints are produced according to forecasted traffic demands, which can be quite different from actual demand, providing reasonable grounds to ignore them in special cases. The latter situation will add more flexibility to our system since every layer will be guided by distinct policies introduced according to the operator's requirements but, unless there is some coordination of policies in the two layers, this may result in an unstable state for the whole system. This means that instability will occur if policy 5.11 is not enforced on DRsM despite the fact that policy 5.10 is enforced on ND. We re-state here that DRsM functionality is distributed at every router operating within a RPC, without a global coordinated view, while ND functionality is centralised and *has* such a global view, so relevant constraints will most probably result in a more stable overall system.

The same principles can be applied to policies driving the behaviour of the Dynamic Route Management Component with the policies influencing the routing decisions of Network Dimensioning. Moreover, SLS subscription management policies may result in the introduction of SLS Invocation Management policies presented also in this chapter due to the same hierarchical relationship between the two aforementioned components.

5.4 Dynamic Resource Management Service Creation through Policies

In this section, we apply the framework/methodology presented in Chapter 3 for creating a management service almost exclusively through policies for the case of Dynamic Resource Management. In Figure 5-10, we show the components that need to be present as well as the relevant policies comprising the management logic that combines the static functionality of the components in order to create a dynamic resource management service. We also present concrete examples of policies which were described in section 5.2.3.2.

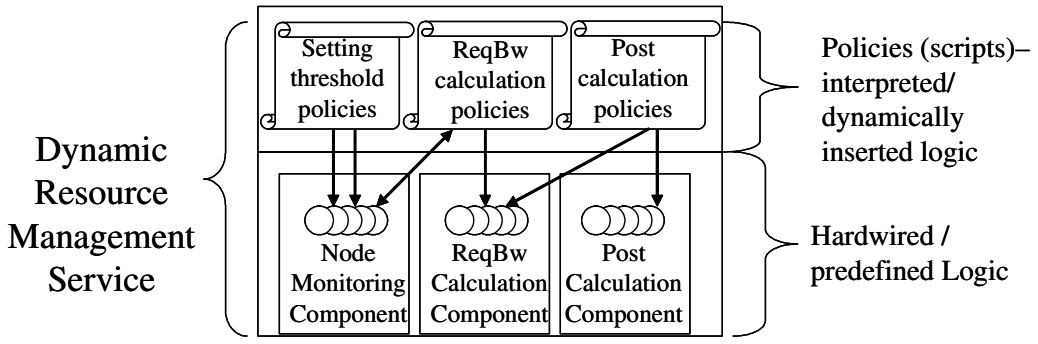


Figure 5-10 DRsM Service Creation through policies

The three basic components that constitute the hardwired logic are the Node Monitoring component, the Required Bandwidth Calculation component and the Post Calculation component. The different managed objects inside every component represent separate functional units and

different strategies for realising the component's functionality, leaving the decision on which strategy to be applied to the policies performing operations on these objects. First *node monitoring* policies define the strategy to be used in order to calculate the upper and lower thresholds in the monitoring components. For example, a policy might state that for the EF PHB the upper threshold should be placed 20% above the current utilisation (Example 5.12). A similar policy could be enforced for the lower threshold as well (Example 5.13). The above policies are triggered every time there is a new bandwidth allocation for the PHB.

Example 5.12 `if PHB ==EF and new_Bw_Allocation == TRUE then Upper_threshold == 20% above Current_Load`

Example 5.13 `if PHB == EF and new_Bw_Allocation == TRUE then Lower_threshold == 20% below Current_Load`

The second type of policy rules required to be enforced are those that define the strategy for calculating the Required Bandwidth. These policies are triggered by upper and lower threshold crossing events issued by the node monitoring component which was previously configured by the node monitoring policies on how to calculate the corresponding thresholds (Examples 5.12-13). For example, a Required Bandwidth calculation policy might state that when there is an upper threshold crossing alarm then the required bandwidth should be calculated by increasing the bandwidth allocated to that PHB by 10%. Again other conditions might hold for the execution of the policies such as time period conditions.

Example 5.14 `if PHB == EF and Upper_Threshold_Crossed == True then Increase Required_Bw by 10%`

Example 5.15 `If PHB == EF and Lower_Threshold_Crossed == True then Decrease Required_Bw by 10%`

Finally, the last type of policies are the post calculation policies and the target component holds the managed objects of the network element (router) which contain different strategies for excess and spare bandwidth treatment. For example, if there is a potential congestion in the node, i.e. the sum of required bandwidths of all the PHBs exceeds the link capacity, then an action should be executed in order to reduce the required bandwidth based on a predefined strategy as previously explained (post processing functions) e.g. equally reduce the required Bandwidths of all PHBs in order to fit the link capacity (Example 5.16). A policy could state that if there is spare bandwidth then an action should be executed that proportionally splits the remaining bandwidth and distributes it to the supported PHBs (Example 5.17). The result of this post calculating function will result in the actual bandwidth value to be allocated to every PHB. Similar policies could be enforced in the case of congestion that allocate the value provided by Network Dimensioning or to equally reduce all the required Bandwidth values so that their sum fits the link capacity. As it

can be observed the conditions of these policies are based on the required bandwidth values calculated according to the previously defined policies.

Example 5.16 `if Link_ExcessBw == True then equally_reduce_Bw`

Example 5.17 `if Link_SpareBw == True then proportionally_distribute_Bw`

Finally, the above policy rules will result in a new allocation of bandwidth that trigger the recalculation of the threshold values as defined by the relevant policy rules (Examples 5.12 and 5.13).

As it can be observed from the above, in order to combine the hardwired functionality of the components, policies need to interact with more than one component e.g. registering events on the node monitoring component and executing actions on the Required Bandwidth Calculation component (Examples 5.14-15). In the case of DRsM, there is a sequential execution of policy rules in order to dynamically create a DRsM service which is achieved by specifying rules that their triggering events and condition hold only after a component has finished its execution. For example, in order to evaluate if there is spare bandwidth that needs to be allocated, the calculation of the required bandwidth function must be completed (Examples 5.14-15) in order for the relevant (post calculating) policies to be enforced (Examples 5.16-17). On the other hand policies that define different treatment to the supported PHBs can be concurrently executed since they do not have this type of relationship. The latter case applies also to the policies defining the upper and lower thresholds (for the same PHB) as shown in the examples 5.12-13 i.e. their actions are executed at the same time since they contain the same conditions.

5.5 Summary and conclusions

In this chapter, we defined the policies that drive the behaviour of every component of the Policy-based QoS management architecture presented in Chapter 4. Following the generic classification of QoS policies described in section 4.8, we presented SLS management and Traffic Engineering policies driving the offline as well as the dynamic management components of both sub-systems. The behaviour of every component was briefly described and a detailed description of the parameters and the specific functionality that can be policy-driven was provided. The policies presented here provide an important contribution in the area of QoS policies since they provide a holistic approach of the management functionality which can be implemented using policies in contrast with the simple configuration policies presented in the literature.

SLS-subscription and invocation management policies influence the admission control decisions at both levels and they are mostly triggered by conditions related to characteristics of SLS

subscriptions and Invocations enabling differential treatment on different types of SLSs. Moreover, SLS subscription policies are also based on the resource availability information provided by the offline Network dimensioning component while SLS invocation policies also take into account the actual status of the network with information provided by network monitoring.

In the same manner, Traffic Engineering policies influence the resource allocation and the routing decisions of the different supported QoS classes at both levels i.e. offline and dynamic. While Network dimensioning policies are based on the different characteristics of the Traffic Matrices which are the basic input of ND, the DRsM and DRtM policies are based on the actual network status provided by both the network and node monitoring components and achieve an adaptive management behaviour regarding decisions on rerouting of currently invoked SLSs and allocation of bandwidth to the PHBs in every link of the network.

We also presented a formal representation of the above policies using the object-oriented Policy Core Information Model (PCIM) and its extensions (PCIMe) as defined jointly by the IETF and DMTF standardisation bodies. For this reason, we extended the core classes provided by the aforementioned models by defining specific classes and their properties needed to represent the SLS management and Traffic Engineering policies.

Two case studies were also presented of the two general frameworks presented in Chapter 3 for applying policies to a hierarchical management system and for creating management services using policies. We focused on the hierarchical relationship of the ND and DRsM components and discussed the situation where an ND policy could possibly be refined into a DRsM policy by giving a specific example. Finally, we showed how we can combine the static functionality of the DRsM component with the relevant policies in order to dynamically create a DRsM service in our system.

All of the work presented in this chapter has been carried out by the author of this thesis and part of it has been published in [Fleg02], [Fleg03], [Trim02a], [Trim02b] and [Trim03b]. In the following chapter, we describe the design and the implementation of the policy management components and present specific examples of the QoS policies described in this chapter showing their transformation from their definition by the operator until their enforcement.

Chapter 6

6 Prototype System Design and Implementation

Chapter 6 of this thesis presents the design and the implementation of our prototype system that has been developed in order to provide a proof of concept validation of the proposed work on policies for QoS management. Specifically, we focus on the policy management sub-system components i.e. the Policy Management Tool, the Policy Repository and the Policy Consumer that need to be deployed over the components of the QoS management architecture presented in Chapter 4 in order to drive its behaviour dynamically through policies. We present specific examples of the enforcement of policies applied to the offline network dimensioning (ND) and the dynamic resource management component (DRsM) components. We show the different way of representation of the policy rules at every stage of their life-cycle i.e. from high-level directives defined in the Policy Management Tool to an object-oriented format in the Policy Repository and finally to interpreted scripts at the Policy Consumer, presenting also their effect on the behaviour of the managed network.

The policy management tool provides a policy creation environment to the administrator who is responsible for introducing policies in the form of high-level rules according to a well defined syntax. As presented in Chapter 2, many policy languages have been presented in the literature, each one addressing the requirements of their application domain. In this chapter, we introduce a high-level policy language that enables the administrator to define and update policies related to QoS Management of IP DiffServ Networks as defined in the previous chapter. The core functionality of the PMT is to translate the policy rules from the high-level representation to an object-oriented form that conforms to a predefined information model/schema in order to be stored to the Policy Repository. Before every policy rule is stored to the repository, validation and consistency checks should be performed in order to ensure the correctness of every policy rule and avoid any conflicting situations that may arise between policy rules. Two types of validation are identified: syntactic validation i.e. checking that the policies conform to the predefined syntax of the language and validation of the semantics of the policy rules, checking also that the actual conditions and actions are compatible with the capabilities of the managed system. The latter type of validation has been addressed by [Lymb04] and has not been studied and implemented in the

context of this work. Conflict detection is the final check performed to every policy rule, which aims to detect inconsistencies that may arise between the newly introduced policy rule and the already existing rules, which are already enforced or ready to be enforced. Recently, conflict detection methodologies have been proposed in the literature such as [Band03] which are based on reasoning techniques and are out of the scope of the work presented in this thesis. Some initial work in this area of conflict detection in the context of the QoS Management policies is presented in [Char05]. Finally, since the policy management tool provides the interface to the human administrator to define policies driving the behaviour of the management system according to his/hers business objectives, a human friendly graphical interface has been implemented and its functionality is presented in this chapter.

The Policy Repository component is responsible for storing the policy rules after they are defined and the appropriate checks have been performed in the PMT. The widely adopted LDAP Directory has been chosen as the technology to realise the Policy Repository component in our prototype implementation. The guidelines produced by the IETF Policy Framework WG [RFC3703] are used to map the technology independent QoS Management Information model presented in the previous chapter to a directory that uses LDAP [RFC2251] as its access protocol. Examples of policy rules are presented in both representations demonstrating how the policy rules are stored in our Policy repository. Finally, the enforcement of policies requires access to the repository by the Policy Consumers present in our architecture.

The Policy Consumer can be considered as the most critical component of the policy management system since it is responsible for the enforcement of the policy rules. In this chapter, we present a detailed design of the policy consumer which is further decomposed into subcomponents describing their functionality that achieve the dynamic introduction of (interpreted) logic that enforce the policy rules defined by the administrator. Two instances of policy consumers are presented for validating our proposed system. The first one is attached to the Network Dimensioning Component and is responsible for implementing the relevant policies presented in the previous chapter. Since Network Dimensioning is an offline component the corresponding policy consumer was implemented by attaching it to the existing implementation of the ND component and enforcement of policy rules require no external interaction with other components. The second instance of the policy consumer presented in this chapter is the one which is responsible for enforcing policies related to the DRsM component. DRsM policies as described previously are executed based on the actual network status and the corresponding policy consumer implementation interacts with a simulated network for receiving notification events and performing the corresponding actions.

The effect of the enforcement of policies that drive the behaviour of the ND and DRsM components are also presented while demonstrating their transformation from the high-level

syntax in the PMT, to LDAP objects in the Policy Repository and finally to interpreted scripts in the Policy Consumer.

The structure of this chapter is as follows.

Section 6.1 presents the implementation of the Policy Management Tool. We focus on the implementation of the policy definition language, describing also the functionality of the graphical user interface that provides a more user friendly environment to the administrator. An LDAP browser has also been implemented that allows the administrator to browse through the policy objects stored in the Policy Repository. Finally, examples of ND related policy rules are presented using our policy language.

Section 6.2 presents the implementation of the policy repository. Since the LDAP Directory was chosen to realise the policy repository we used an open source implementation of the Directory and we present a CORBA-based implementation of the LDAP protocol to ease the interaction with the various components of our system. Examples of policy rules are also presented showing their mapping to LDAP objects from the representation using the information model presented in the previous chapter based on the PCIM and PCIMe.

Section 6.3 describes the detailed design and implementation of the policy consumer. Two instances of the policy consumer are presented. The first operates on managed objects of the network dimensioning component following the tightly coupled approach as presented in Chapter 3. The second operates on the managed objects of the DRsM component which has been implemented inside the NS network simulator and affects the managed objects of the simulated network nodes. Examples of the interpreted logic in pseudo-code are also presented produced by the policy consumer for enforcing the aforementioned ND policies and their effect on the network behaviour is also demonstrated in a monitoring tool. Examples of DRsM policies are also presented using our policy language and their effect is demonstrated by measurements taken during the simulation-based experiments.

Finally, Section 6.4 summarises what has been presented in this chapter highlighting the relevant contributions.

6.1 Policy Management Tool

A high-level definition language has been designed and implemented that provides to the administrator the ability to add, retrieve and update policies in the Policy Repository. The administrator enters a high-level specification of the policy, which is then passed to a translation function that maps this format to entries in an LDAP Directory realising the Policy Repository (see next section) through LDAP *add* operations, according to an LDAP schema of our

information model; the latter has been produced following the guidelines described in [RFC3703]. The format of a policy rule specification is shown below:

[Policy ID] [Group ID] [time period condition [-r]] **[if** {condition [-r][and] [or]}] **then** {action [-r] [and]} [role]

The first two fields define the name of the policy rule and the group that this policy belongs to so that the generated LDAP entries should be placed under the correct policy group entry. The time period condition field specifies the period that the policy rule is valid and supports a range of calendar dates, masks of days, months as well as range of times. There are three different formats that a time period condition can be specified:

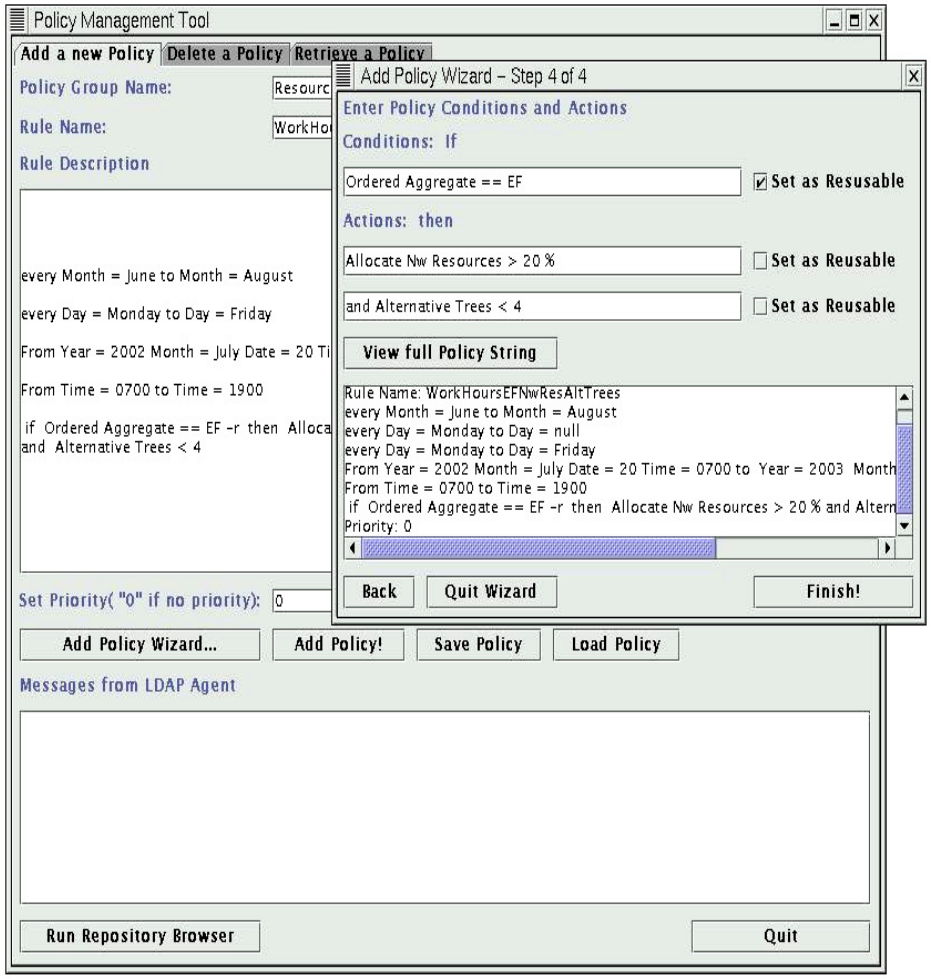
- "Every attribute=value & | to attribute=value", where attribute = Day | Date | Month. When the attribute is Day the value can be Sunday | Monday | … | Saturday. When the attribute is Date the value can be 1 | 2 | … | 31. When the attribute is Month the value can be January | February | … | December.
- "From Time=value to Time=value", where value is of the form hhmm.
- "From {attribute=value} to {attribute=value}", where attribute = Year | Month | Date | Time. When the attribute is Year, the value is of the form yyyy. When the attribute is Month, the value can be 01 | 02 | … | 12. When the attribute is Date the value can be 01 | 02 | … | 31 and when the attribute is Time, the value is of the form hhmm.

The following {if then} clause represents the actual policy rule where the condition and action fields are based on the information model described in Chapter 5. Compound Policy Conditions are also supported both in the Disjunctive Normal Form (DNF) (an ORed set of ANDed conditions) and in the Conjunctive Normal Form (CNF) (an ANDed set of ORed conditions) as well as Compound Policy Actions representing a sequence of actions to be applied. Our implementation also caters for the notion of rule-specific and reusable conditions and actions in a way that every time a new policy rule is added, it first checks if its conditions and actions are already stored in the repository as reusable entries. If such entries exist, an entry is added with a Distinguished Name pointer to the reusable entry under the policy rule object while if not they are treated as rule-specific, placing the condition entry below the policy rule entry. Moreover, when the administrator adds a rule, he/she can optionally indicate that some parts of this rule (conditions and/or actions) should be treated as reusable entities by adding a "-r" identifier at the end so that the relevant entries in the repository will be stored in a specific location that later could be referenced by other policy rules which contain the same entities.

A compiler has been implemented in order to parse and translate the policy rules specified with the above syntax using SableCC [SableCC]. This is an object-oriented framework that generates

compilers 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. Translation classes have been implemented that map the introduced policy rules to LDAP *add* operations, according to an LDAP schema of our information model presented in the previous chapter. The policy language specification that contains the lexical definitions using regular expressions and the grammar productions in Backus-Naur Form is included in Appendix A. Examples of how policies are stored in the repository are presented in the next section.

Notifications to the corresponding Policy Consumers are also sent by the PMT every time a policy rule is successfully added to the repository. The role attribute of every policy rule is used to distinguish which policy consumer to notify since this attribute represents the properties of the target component that each Policy Consumer manages according to the Policy Core Information Model (PCIM) [RFC3060].



(a)

(b)

Figure 6-1 a) Policy Management Tool Screenshot b) **LDAP Browser Instance**

A Graphical User Interface (GUI) has been implemented to assist the administrator in managing policies in the system (Figure 6-1a). It provides capabilities to add new policies, retrieve or delete existing policies. The addition of the policies can either be done directly by writing the whole policy rule in the format described previously or follow a wizard that guides the administrator through the process of creating and entering a new policy to the system step by step. In order to retrieve a policy rule from the repository, the administrator may optionally specify the group name, under which the requested policy is placed, and provide the name of the policy rule to be retrieved. In case the provided rule name exists more than once in the repository, they are notified accordingly and therefore should try again by specifying the rule's group name or even the group path. In the same manner, when the administrator wishes to delete a policy rule, he/she should provide the name of the policy rule to be deleted and optionally the group name that this policy belongs to. In case the policy rule that is being deleted has reusable conditions or actions, the entries that reside below the rule and contain DN pointers to the reusable entries are deleted, but the reusable conditions or actions that reside below the repository entry are not.

A directory browser has also been implemented (Figure 6-1b) and integrated with the tool that enables the operator to browse the information stored in a tree-structured repository like the

LDAP Directory. In the browser's design, the idea of a current entry was adopted for displaying the contents of a tree. Each browser instance has one current entry, which is a selected tree entry. Its distinguished name (DN), its attributes' types and values, its superior entry's distinguished name and its subordinate entries' relative distinguished names (RDNs) are displayed. The current entry's position in the tree can be identified by the current entry's distinguished name. The user may select any of the current entry's subordinate entries or its superior entry and make it the browser's current entry. This way, one can move up and down the hierarchy of the Directory Information Tree (DIT) that is accessed. If an attribute's entry is a pointer to another entry, the user can make it the current entry, in which case they still keep the option to go back. In addition, the user is allowed to fill in the DN of the entry to be displayed in the Current Entry field. More screenshots of the PMT GUI are presented in Appendix B.

6.1.1 Examples of Network Dimensioning Policy definitions

In order to demonstrate the results of the enforcement of ND policies we used a 10-node (nodes 0-9) 36-link random topology and a traffic load of 70% of the total throughput of the network shown in Figure 6-2. We defined as the *total throughput* of a network the sum of the capacities of the first-hop links emanating from all edge nodes. This is the upper bound of the throughput, and in reality it is much greater than the real total throughput a network can handle. This happens because, although the sum of the first-hop link capacity imposes this limit, the rest of the backbone might not be able to handle that traffic.

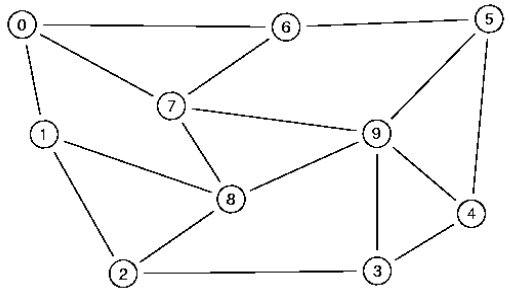


Figure 6-2 Topology used for the policy examples

Our first example (P1) concerns a policy rule that wants to create an explicit LSP following the nodes 4, 9, 7, 6 with the bandwidth of the Traffic Trunk being 2 Mbps that is associated with this LSP. The LSP will be established when a traffic matrix is received that contains an entry that satisfies the conditions of the policy rule. The policy rule is entered with the following syntax:

Policy P1: `If TT_QC==EF and TT_Ingress==4 and TT_Egress==6 then Setup LSP 4-9-7-6 2000 kbps`

The second example (P2) of a policy rule defines the behaviour of the network dimensioning component regarding the optimisation objective based on which the network configuration is calculated. As mentioned in section 5.2.2, this concerns the effect of the cost function exponent in the capacity allocation of the network where by increasing the cost function exponent, the optimisation objective that avoids overloading parts of the network is favoured. So, if the administrator would like to keep the load of every link below a certain point then he/she should enter the following policy rule in our system using again our policy notation:

Policy P2: `If maxLinkLoad > 80% then dimension with minimising MaxLinkLoad objective`

The above policy rule causes network dimensioning to calculate a new configuration with a different value of the cost function exponent if the maximum link load of the produced configuration is above 80%. The detailed description of how these rules are actually enforced by the policy consumer is described later in the relevant section.

6.2 Policy Repository

6.2.1 Introduction to the LDAP Directory Service

In our prototype implementation, we use the LDAP directory as our policy repository. A directory is a specialized database optimised for reading, browsing and searching. Directories tend to contain descriptive, attribute-based information and support sophisticated filtering capabilities. Directories generally do not support complicated transaction or rollback schemes found in database management systems designed for handling high-volume complex updates. They may have the ability to replicate information widely in order to increase availability and reliability, while reducing response time.

There are many different ways to provide a directory service. Different methods allow different kinds of information to be stored in the directory, place different requirements on how that information can be referenced, queried and updated, how it is protected from unauthorized access, etc. Some directory services are local, providing service to a restricted context (e.g., the finger service on a single machine). Other services are global, providing service to a much broader context (e.g., the entire Internet). Global services are usually distributed, meaning that the data they contain is spread across many machines, all of which cooperate to provide the directory service. Typically a global service defines a uniform namespace, which gives the same view of the data no matter where you are in relation to the data itself. The Internet Domain Name System (DNS) is an example of a globally distributed directory service.

LDAP stands for Lightweight Directory Access Protocol [RFC2251] and it provides services that act in accordance with the X.500 recommendation [X500]. The LDAP protocol model conforms to the functional model of the X.500 OSI Directory (Figure 6-3). According to this model LDAP servers (Directory Service Agents) contain the information and are accessed by LDAP clients (Directory User Agents).

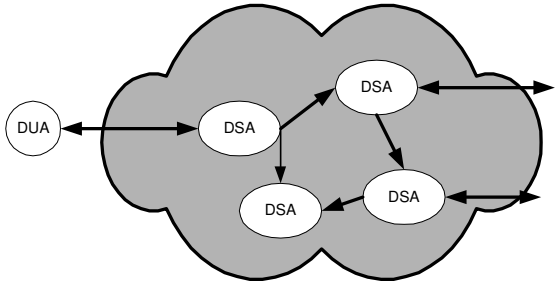


Figure 6-3 The Directory Functional Model

An LDAP client sends a protocol request and a description of the operation to be performed. The server then performs the operation on behalf of the client and returns a response containing the results of the operation or any possible errors. This response may contain referrals to other LDAP servers but this is supported only in LDAPv3.

The *Data Model* of the LDAP protocol defines a *Directory Information Tree (DIT)* which is jointly administered by one or more LDAP servers. The DIT is made of *entries* which are made of a collection of *attributes*. Each entry has a name which uniquely identifies it and is called *Distinguished Name* (DN). The *LDAP schema* provides a listing of all classes and attributes from which entries are derived. Every new entry in the DIT must conform to the definitions of this schema.

The Protocol Data Unit defined by LDAP is the LDAPMessage. An LDAPMessage envelopes all protocol operations and contains common fields that are required in all protocol exchanges. The LDAPResult is the construct used by the protocol to return any success or failure of an LDAP operation. The most important LDAP operations are briefly mentioned below:

- *Search Operation:* A SearchRequest is sent by a client to a server to perform a search operation on its behalf. A SearchResponse is sent by the server after the search operation is finished.
- *Modify Operation:* This operation allows a client to request that a modification of an entry be performed on its behalf.
- *Add Operation:* A client is allowed to request the addition of a new entry to the DIT.

- *Delete Operation:* The removal of an entry using a client DeleteRequest is possible with this operation.

All these operations and other more specific along with the definitions of data types are in detail described in [RFC2251].

6.2.2 CORBA-based access to the LDAP Policy Repository

In our prototype implementation, we used the OpenLDAP [openldap] software which is an open source implementation of the LDAP Directory, providing APIs that enable users to write applications to access, manage, update, and search for information stored in directories accessible using LDAP in many programming languages such as C/C++, Java, python and perl. Since our system architecture described in chapter 4 is a large scale distributed system, it is valid to consider CORBA [CORBA] as the technology to support the remote interactions between the components. This was the key motivation for mapping the LDAP functionality to CORBA realizing the Policy Repository as an LDAP Directory offering a CORBA IDL interface, identical to the LDAP specifications [RFC2251], to the rest of the components. The following Figure 6-4 shows the part of the specification of an LDAP server in Interface Definition Language (IDL) providing a LDAP search operation to every LDAP client in our system. The specification of the rest of the LDAP operations in IDL can be found in Appendix C. Note that the CORBA implementation of the LDAP search operation returns the result in a single message while the LDAP protocol returns the multiple matching entries in a series of messages, one for each entry. The results are terminated with a result message, which contains an overall result for the search operation.

```
typedef string LDAPDN_t;
enum Scope_t {
        sc_baseObject,
        sc_singleLevel,
        sc_wholeSubtree
    };
typedef string Filter_t; // filter for this implementation
struct SearchResultEntry_t_struct {
        LDAPDN_t                objectName;
        AttributeList_t         attributes;
    };
typedef SearchResultEntry_t_struct SearchResultEntry_t;
typedef sequence<SearchResultEntry_t> SearchResultEntryList_t;

interface LDAPServer {
   void Search (
        in  LDAPDN_t                    baseObject,
        in  Scope_t                     scope,
        in  Filter_t                    filter,
        in  AttributeDescriptionList_t  attributeTypes,
        out SearchResultEntryList_t     searchResultList
    )   raises   (noSuchObject,   invalidDNSyntax,   invalidFilterSyntax,
generalError);

}; // interface LDAPServer
```

Figure 6-4 LDAP Search operation in IDL

In the following section, we describe the way policy rules are stored in the policy repository, showing how policies represented in the information model described in Chapter 5 are mapped to LDAP entries in the Directory.

6.2.3 Examples of ND policies as LDAP entries in the Policy Repository

Since our policy information model presented in Chapter 4 is based on the PCIM and PCIMe, we followed the guidelines described in [RFC3703] and [Pana04] in order to map our model to a form that can be implemented in a directory that uses LDAP as its access protocol. In a LDAP schema, three types of classes can be defined: abstract, structural and auxiliary. An abstract class is used to derive other classes, providing common characteristics of such object classes. Structural classes are defined for use in the structural specification of the DIT and are used in the definition of the structure of the names of the objects for compliant entries. Auxiliary classes may be used in the construction of entries of several types. In certain environments, there is a need to be able to add to or remove from the list of attributes permitted in an entry of a particular, perhaps standardized, class. This requirement may be met by the definition and use of an auxiliary object class having semantics, known and maintained within a local community, which change from time to time as needed.

Two types of mappings of the information model classes to LDAP schema classes are involved. For the structural classes in the information model, the mapping is basically one-to-one where information model classes map to LDAP classes and information model properties map to LDAP attributes. The information model relationship classes can be mapped to LDAP auxiliary classes, to attributes representing distinguished name (DN) references and to superior-subordinate relationships in the DIT. We present below, the ND policy examples presented in the previous section showing how they are represented using the classes defined in the previous chapter and describing their mapping to LDAP entries stored in the repository.

In Figure 6-5 the policy rule P1 (**If** TT_QC==EF and TT_Ingress==4 and TT_Egress==6 **then** Setup LSP 4-9-7-6 2000 kbps) is modelled using the classes based on PCIM and PCIMe classes. As it can be seen, it comprises a compound policy condition which represents a combination of 3 simple policy conditions in Disjunctive Normal Form (ORs of ANDs) each of them belonging to the same Group (GroupNumber =1) and a Policy Action represented by the TENDPolicySetupLSPAction class derived from the PolicyAction abstract class defined in PCIM. The first simple policy condition uses a QC variable which takes integer values from 1 to 4 (EF is 1, AF1x is 2, etc), the second and third simple policy conditions use an Ingress node and Egress node variables which take integer values form 0 to 9 for our network topology (the egress node

simple policy condition is not shown in Figure 6-5 for illustrative purposes). The aggregations used in order to define this rule are also depicted as in PCIMe.

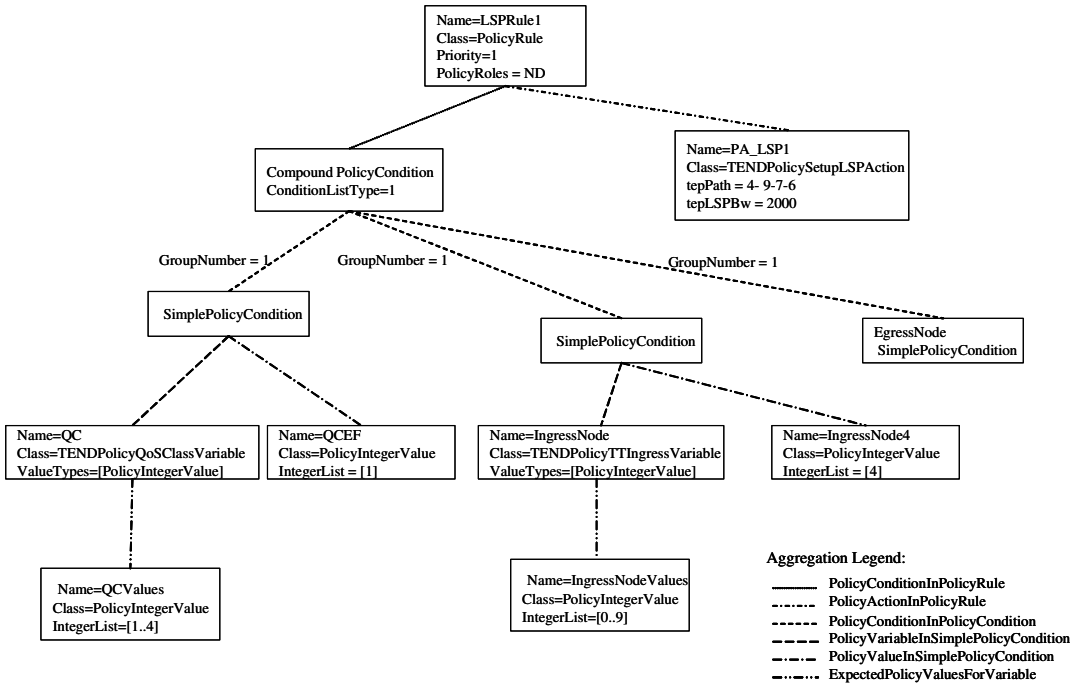


Figure 6-5 Policy Rule P1 modelled according to PCIM/PCIMe

In Figure 6-6 the above policy rule is presented as LDAP entries in our Policy Repository. The PolicyRule Class is mapped to the pcelsRuleInstance structural class. The structural classes pcelsConditionAssociation and pcelsActionAssociation represent the association of the compound policy condition and the policy action to the policy rule. The actual compound condition and action are auxiliary attachments to these structural classes. Therefore, the rule specific action and compound condition are subordinated (DIT contained) to the rule entry. The conditions are tied to the compound condition in the same way the compound condition is tied to the policy rule. The simple condition, variable and value classes are all mapped to auxiliary classes which are directly attached to the pcelsConditionAssociation structural class.

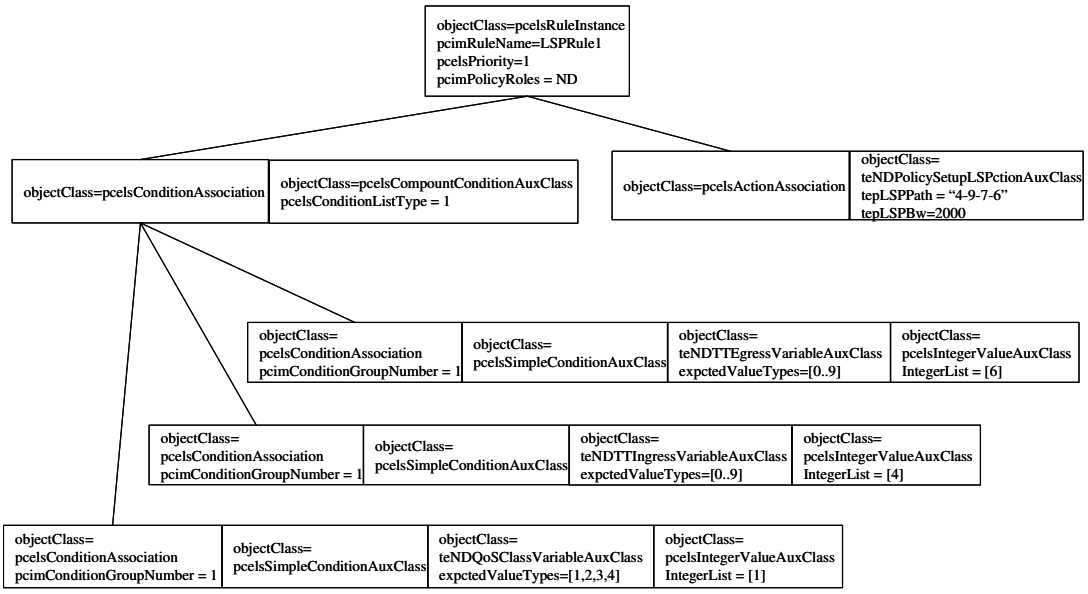


Figure 6-6 Policy Rule P1 mapped to LDAP entries

Using the same methodology, policy rule P2 (**If** maxLinkLoad > 80% **then** dimension with minimising MaxLinkLoad objective) is modelled according to PCIM/PCIMe using a SimplePolicyCondition using the TENDPolicyMaxLinkLoadVariable class and the TENDPolicyOptimisationObjectiveAction class derived from the Policy Action abstract class as defined in the previous chapter. These classes are mapped to structural and auxiliary classes as described previously for policy rule P1.

The above representation of policy rule P1 treats the policy conditions and actions as rule specific. As mentioned in section 6.1, our implementation of the Policy Management Tool caters for reusable components in the policy rule. For example, the Simple Policy Condition which defines the QoS Class is highly likely to be used in many policy rules since all the relevant ND policy actions that should be applied when ND calculates the configuration for this type of traffic will contain this policy condition. In this case the auxiliary condition classes are not attached to the structural condition association class. Instead, the latter class contains a DN reference to a structural class that is placed under a reusable container entry. The actual condition is attached to this structural class and because it is named under an instance of the container entry is clearly identified as reusable.

In order to show the benefits of using reusable conditions or actions in policy rules, we performed a test, checking the amount of memory consumed by the LDAP server. Two cases were considered. In the first case, we defined only policies with rule specific components and measured the amount of memory allocated by the LDAP server with respect to the number of policies. In the second case, we defined only policy rules that contain reusable conditions and actions which are already stored in the repository, creating this time only entries with a pointer to the reusable

one as described previously for every condition and action. For our measurements, we used policies that contain 3 simple policy conditions and one policy action such as policy P1 and we varied the number of policy rules from 1 to 50.

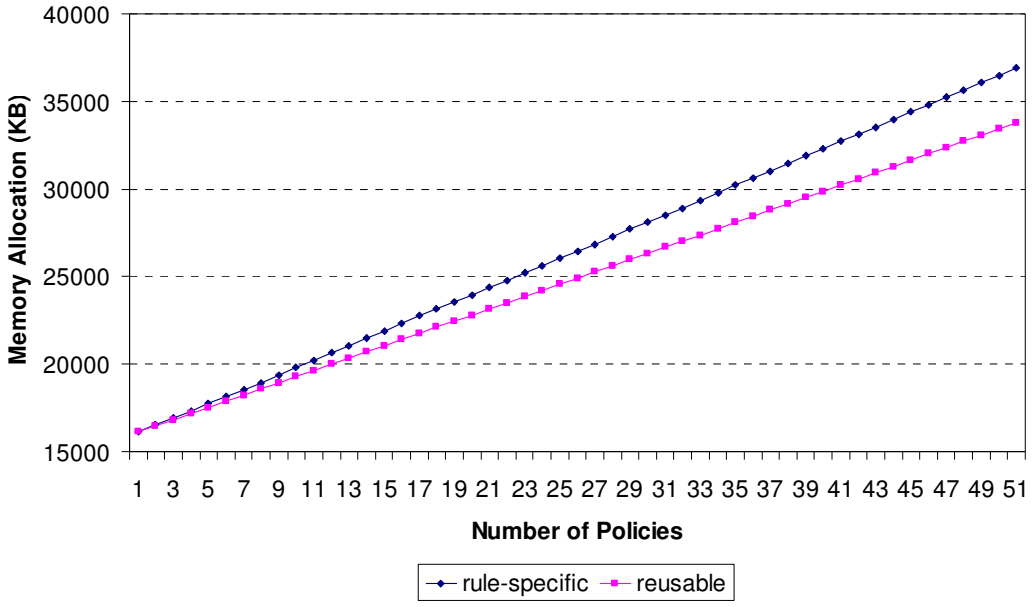


Figure 6-7 LDAP Server Memory Allocation of rule-specific vs reusable components in policies

Figure 6-7 depicts the memory allocation of the policy repository in both cases described above. As it can be clearly observed, in the case of policy rules with reusable components there is an increasing gain in memory allocation as the number of stored policy rules increase. The result presented in this figure is easily predictable since the gain introduced by the use of reusable components is a linear function of the memory allocated by policies with rule specific components as shown in the equation below.

$$G_i = 0.075 * M_i^{rule-spec} \qquad \text{(Eq.2)}$$

where i is the number of policies, G_i the gain introduced by the use of reusable components for i stored policies and $M_i^{rule-spec}$ is the memory allocated by i number of policies with rule specific components.

6.3 Policy Consumer

Policy Consumers may be considered as the most critical components of the policy management framework since they are responsible for enforcing the policies on the fly while the system is running. In the following Figure 6-8, a decomposition of the Policy Consumer component is

depicted and its interactions with the other components of the architecture. A similar approach was presented in [Marr96] for implementing a policy manager-agent.

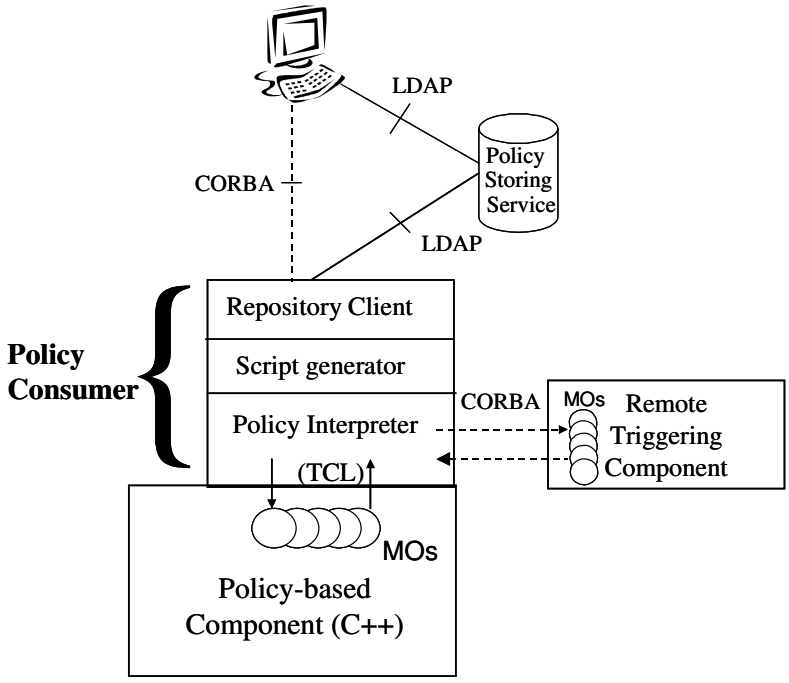


Figure 6-8 Decomposition of a Policy Consumer

The key aspect of policies apart from their high-level declarative nature is that they can also be seen as a vehicle for "late binding" functionality to management systems, allowing for their graceful evolution as requirements change. So, a policy capable system should provide the flexibility to add, change or remove management intelligence while according to traditional management models, management logic is of static nature, parameterised only through Managed Objects (MOs) attributes and actions. In order to achieve such functionality, a policy is eventually translated to a script "evaluated" by an interpreter, with actions resulting in management operations. This approach has also been followed by the Management by Delegation (MbD) paradigm [Yemi91] and the IETF Script MIB [RFC2592] specifying an implementation architecture. Policy Consumers can be attached to the component they are associated with, having local access to the MOs or they can enforce the policies remotely. The former approach is depicted in Figure 6-8.

As shown in Figure 6-8, the Policy Consumer component is decomposed in three parts. The first part, the *Repository Client* provides access to the Policy Repository (PR), and is responsible for downloading the associated objects stored in the repository that comprise the policy rules this specific policy consumer should enforce in order to influence the behaviour of the component it is attached to. The repository client has knowledge of how the policy information is stored in the PR. After a new policy is stored, it receives a notification from the Policy Management Tool that

a new policy for this consumer is stored in the repository by passing the appropriate information i.e. the Distinguished Name of the "root" policy object; it then goes and retrieves all the necessary entries from the repository. This part is the same for every instance of the Policy Consumer in our architecture.

The second part of the Policy Consumer is the *Script Generator*, which is responsible for creating the script that implements the policy. It contains logic, specific to the component that the policy consumer is attached to, that automates the process of generating a script from the higher-level representation of policies as they are stored in the repository. The *Policy Interpreter* provides the "glue" between the policy consumer and the policy-based component and interprets a language, which includes functions that perform management operations. Two kinds of functions are identified: those that provide access to local MOs and those that provide access to MOs of remote components. Since a policy rule is defined as a set of actions when certain conditions occur, the purpose of accessing MOs is twofold. First, evaluating the condition of a rule can be done either by polling attributes of the MOs or following an event-driven paradigm where the MOs send a notification that triggers the execution of the policy action. These MOs can be either local or remote, depending on whether the condition of the policy is based on the state of the component that this policy consumer is attached or on information that can be available from a remote component. Second, policy actions are implemented by executing scripts that result in setting attributes or invoking operations that are available at the object boundary of the local MOs.

An important consideration regarding the implementation of a Policy Consumer is related to the choice of the relevant scripting language. Since in our architecture most of the components are implemented in C++, the first important requirement for the scripting language is to be extensible and able to interface easily with C++. Tcl was chosen for this purpose due to the ease with which it interfaces to C++. The interface between Tcl and the C++ environment of the component is in two directions: from Tcl to C++, for accessing local or remote managed objects and from C++ to Tcl for sending notifications and starting the execution of the policy scripts. In order to support remote invocations on MOs, a CORBA environment is used.

6.3.1 Examples of the Enforcement of ND policies

After the P1 rule is correctly translated and stored in the repository, the Policy Management Tool notifies the Policy Consumer associated with ND that a new policy rule is added in the repository, for which then the policy consumer goes and retrieves all the objects associated with this policy rule. From the policy objects the consumer generates code that is interpreted and executed on the fly, representing the logic added in our system by the new policy rule. In our implementation, we have chosen TCL as the scripting language due to the ease with which it interfaces with C, since

the ND component is implemented in C. The pseudo code of how the above policy is realised by the Policy Consumer is shown in Figure 6-9.

```
TT_OA: the set of TTs belonging to OA
For each tt_i ϵ TT_OA we get the following:
  v_ingress, v_egress : ingress, egress nodes
  B(tt_i): bandwidth requirement of tt_i
for each tt_i ϵ TT_EF do
  If ((v_ingress == 4) and (v_egress == 6))
        add_LSP ("4-9-7-6", 2000)
        b(tt_i) = b(tt_i) – 2000
  Else
        Policy not executed – TT not found
```

Figure 6-9 Pseudo-code produced for enforcing (P1)

As it can be seen from the above pseudo-code, it first searches for a TT in the traffic matrix that matches the criteria specified in the conditions of the policy rule regarding the OA, the ingress and egress node. If a TT is found, then it executes the action that creates an LSP with the parameters specified and subtracts the bandwidth requirement of the new LSP from the TT in the traffic matrix file so that the ND algorithm will run for the remaining resources. Note that if the administrator had in mind a particular customer for this LSP then this policy should be refined into a lower level policy enforced on the DRtM component, mapping the address of this customer onto the LSP.

The same procedure explained in the previous example is followed again for P2 and the policy consumer enforces this policy by generating a script, which is shown in Figure 6-10.

```
maxLinkLoad: the maximum link load utilisation
after the end of the optimisation algorithm
n: the cost function exponent (initially = 1)
Optimisation_algorithm n
While (maxLinkLoad > 80)
   n = n+1
   optimisation_algorithm n
```

Figure 6-10 Pseudo-code produced for enforcing (P2)

As it can be observed from Figure 6-11, the enforcement of the policy rule caused the optimisation algorithm to run for 4 times until the maximum link load utilisation at the final step

drops below 80%. The exponent value of Eq.1 presented in Chapter 5 that achieved the policy objective was $n = 4$.

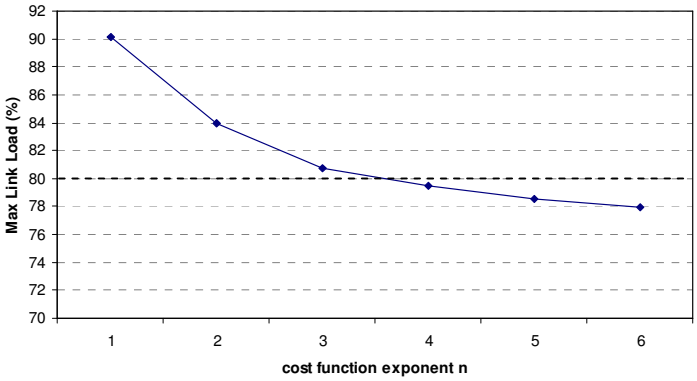
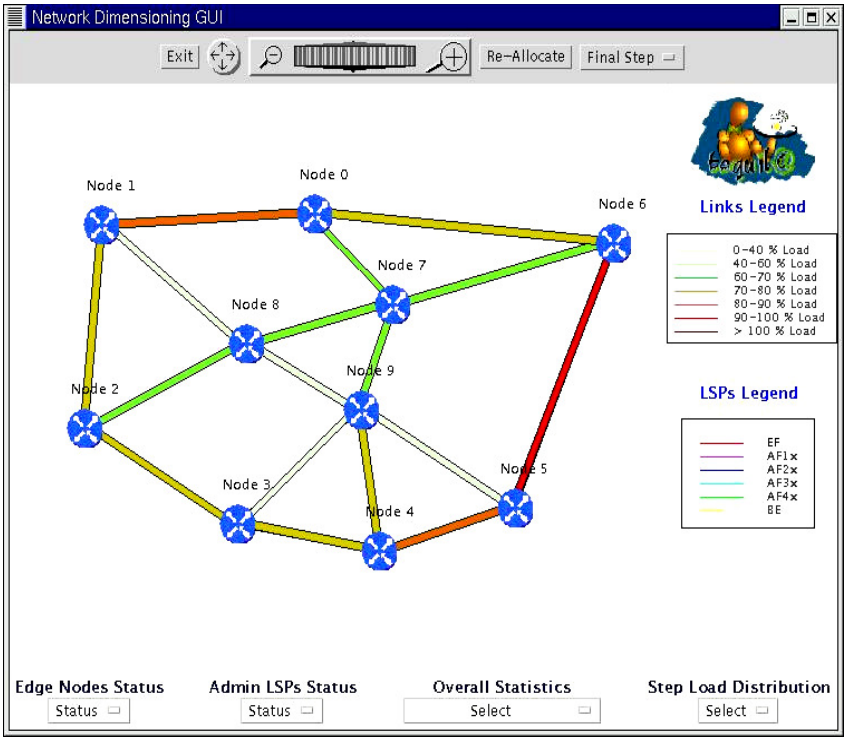
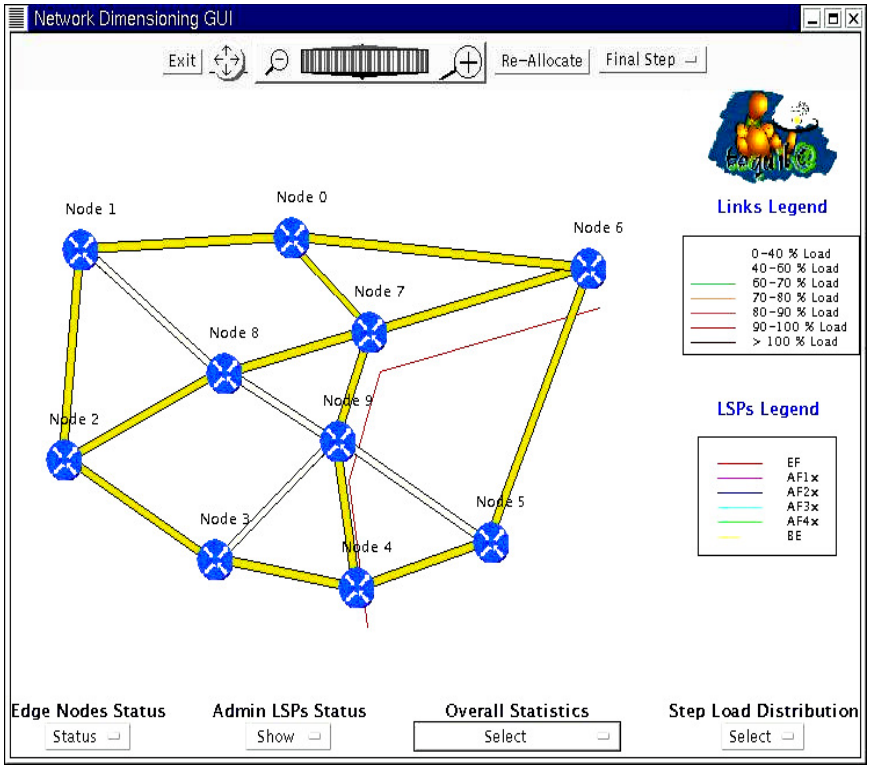


Figure 6-11 Effect of the cost function exponent on the maximum link load utilisation

For the purpose of demonstrating the effects of the enforcement of policies in our system we implemented a TE-GUI shown in Figure 6-12. It depicts the topology of the network that the ND component is calculating a new configuration. The GUI draws the links of the topology with different colours according to load utilisation and all the LSPs for every OA created. It has also the capabilities to display overall statistics for the load distribution for every link per OA as well as statistics for every step of the ND algorithm i.e. average link utilisation, link load standard deviation, max link load, running time etc. In the following figure, two snapshots of the TE-GUI are depicted one before and one after the enforcement of the above policies. As it can be seen from the following figure the enforcement of the policies caused the link load to fall under 80 % (before the enforcement of policies the link 5->6 was loaded over 90%) as well as the LSP created by the P1 is also drawn.



(a)



(b)

Figure 6-12 TE-GUI snapshots (a) before and (b) after the enforcement of policies P1 and P2

6.3.2 Implementation of Policy-based Dynamic Resource Management

In order to evaluate and experiment with the DRsM policies we have deployed our system over a simulated network using NS (Network Simulator) [NS]. In order to implement these policies in the "nodes" of the NS simulator, we actually run the policy consumer remotely (i.e. outside NS) and send the produced scripts to be executed inside the NS simulator as described below.

A tool was implemented that can dynamically enforce policies as scripts while the network simulation is running. NS has a TCL interface to its C++ core implementation, providing us already with the functionality we want to interpret policy scripts on the fly. In order to connect to NS the rest of the components of the policy management sub-system, a CORBA interface was built to NS that enables getting information from it e.g. monitoring data used for triggering the execution of policies. In addition, it allows to "push" scripts to it to be executed realising the policy actions. That way, the scripts are uploaded to NS, which checks a queue and if it finds a new policy script retrieves it and executes it. Similar functionality can be achieved in the opposite direction, with NS putting strings to channels that can be retrieved from a CORBA client remotely at run-time. The implementation architecture of the CORBA interface to NS is depicted in the following figure.

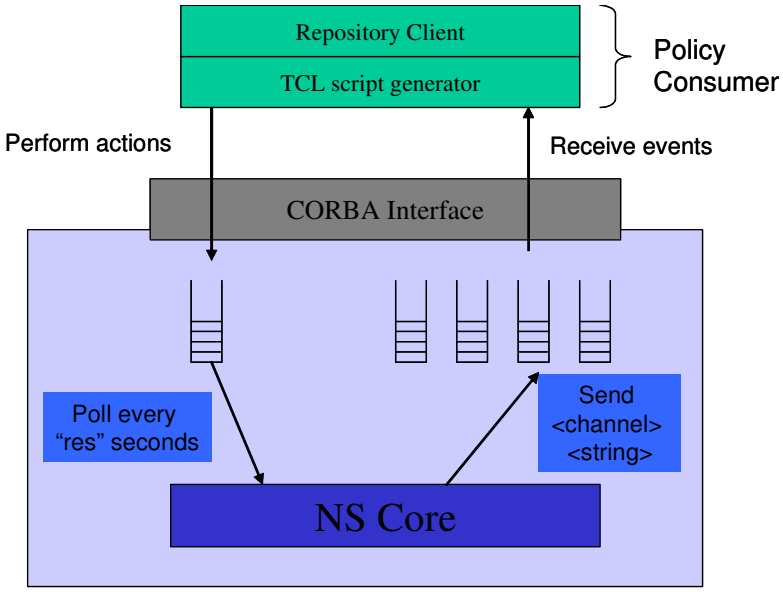


Figure 6-13 CORBA interface to NS

For the purposes of experimenting with the DRsM policies, we used the topology depicted in Figure 6-14 in our simulation model. A traffic aggregate with two classes (EF, AF11) is transmitted to the destination node D via A. The capacity of the links from the individual sources to node A is 100Mbps while the bottleneck link between node A and D is configured to be DiffServ-capable, with total capacity of 10 Mbps. Additionally, we also restricted the EF and total

AF (AF1-4) maximum rates to 5Mbps. Of course when the DRsM policies are enforced the maximum rate varies in the way specified by the rules.

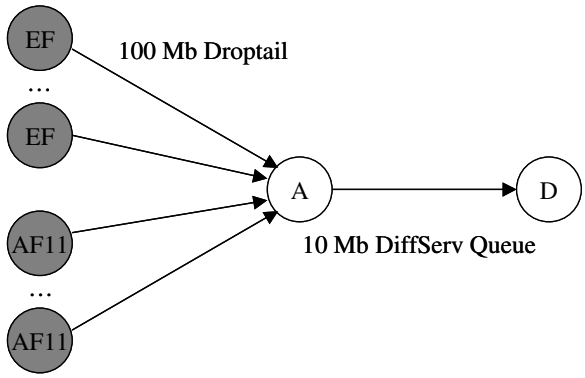


Figure 6-14 Topology used for simulation experiments

In order to demonstrate the effect of the DRsM policies, we enforced the following set of policy rules. First, we introduced policies that set the upper and lower thresholds of the two supported classes as 20 % above and below the current load respectively (P3, P4) every time there is a new bandwidth allocation for the PHB that the alarm was crossed.

Policy P3:

```
if
    (PHB == EF and ( Upper_Threshold_Crossed == TRUE or
    Lower_Threshold_Crossed == TRUE ) and new_Bw_Allocation == TRUE)
then
    Upper_Threshold == 20% above Current_Load and Lower_Threshold == 20%
    below Current_Load
```

Policy P4:

```
if
   (PHB == AF11 and ( Upper_Threshold_Crossed == TRUE or
   Lower_Threshold_Crossed == TRUE ) and new_Bw_Allocation == True
then
   Upper_threshold == 20% above Current_Load and Lower_Threshold == 20%
   below Current_Load
```

Then policies were defined stating that when a threshold is crossed the required Bandwidth is calculated relatively to the offered load i.e. 20% for both classes (P5, P6).

Policy P5:

```
if Upper_Threshold_Crossed == True then Increase Required_Bw by 20%
```

Policy P6:

```
if Lower_Threshold_Crossed == True then Decrease Required_Bw by 20%
```

In the case of spare or excess bandwidth after the required bandwidth calculation, policies that distribute the spare bandwidth (P7) or reduce the calculated value proportionally (P8) to the supported classes were introduced.

Policy P7: `if Link_SpareBw == True then proportionally_distribute_Bw`

Policy P8: `if Link_ExcessBw == True then proportionally_reduce_Bw`

We conducted two sets of experiments. First, we provided 2 on/off data sources for each PHB where initially one EF traffic and both of the two AF11 traffic sources are active and last for 2 seconds, and after that the other EF source starts sending data while one of the AF11 source becomes silent (simple scenario) shown in Figure 6-15.

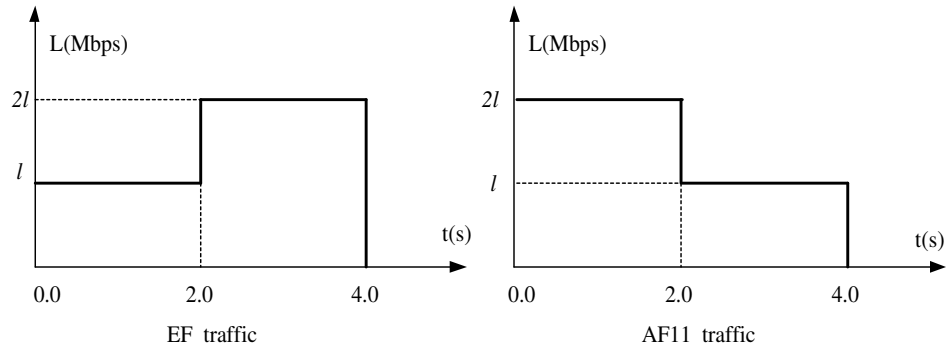


Figure 6-15 Simple scenario case traffic model for DRsM experiments

In the second scenario (complex scenario) shown in Figure 6-16, four on/off data sources are associated with each of the two PHBs and from time 0 one new EF source becomes active every second till time point 3.0 and then starting from time point 4.0 they stop sending data one by one with 1 second of interval till all of them are inactive at time point 7.0. On the other hand, all the AF11 sources are initially active and one of them becomes silent every second till time 3.0. From time point 4.0, one by one they start sending data with the time interval of 1 second and finally at time 7.0 all the AF11 sources return again to active.

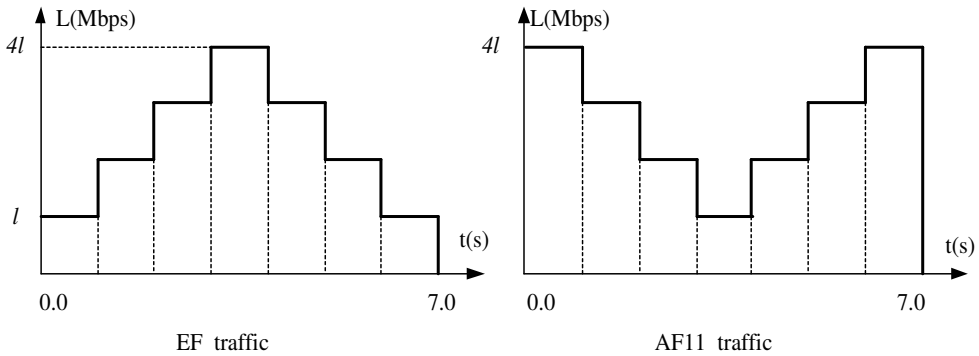


Figure 6-16 Complex case traffic model for DRsM experiments

In Table 6-1 and Table 6-2, we demonstrate the operation of the DRsM policies (P3-P8) with a total aggregate traffic (AF+EF traffic) of 90% for both PHBs for the simple scenario described above (Figure 6-15) i.e. the rate of each data source is set to 3Mbps. We show the events/conditions that trigger the execution of the policy actions of the policy rules presented previously as well as all the values that are calculated by the DRsM components. First, the upper and lower threshold crossing events are depicted that are received by the Policy Consumer and trigger the execution of the policy actions of the rules P5 and P6, calculating the required bandwidth, i.e. 20% above/below the offered load. Note that in this example when a threshold is crossed, the required bandwidth is calculated for all the PHBs. After the required bandwidth is calculated, if there is spare bandwidth (Bw Event), the action of policy rule P7 is executed, distributing the spare bandwidth proportionally to the calculated values of required bandwidth. If there is excess bandwidth, the policy action of P8 is enforced reducing the values of the required bandwidth of the PHBs proportionally in order to fit the link capacity i.e. 10Mbps in our scenario. After the bandwidth is allocated the lower and upper threshold values for the PHB that issued the alarms are recalculated by setting them 20% above and below the current load, as defined by the policies P3 and P4.

For example, at time 2.2077 the upper threshold for the EF PHB has been crossed. This event will trigger the calculation of the required bandwidth for both PHBs by increasing it 20% above the corresponding offered loads (P5) i.e. the required bandwidth for EF is 1.2*5.5993 = 6.7192 while for the AF11 PHB is 1.2 * 3.4007 = 4.0808. These two values of required bandwidth will trigger the enforcement of policy rule P8 since their sum is above the link capacity (10Mbps) i.e. 6.7192+4.0808 = 10.8 (excess bandwidth is 0.8 Mbps). In order to calculate the value of the bandwidth to be allocated for every PHB according to P8, the required bandwidth should be reduced proportionally to fit the link capacity e.g. for the EF PHB 6.7192-(6.7192/10.8) * 0.8 = 6.2213. After the bandwidth is allocated the upper and lower thresholds for the EF PHB are again calculated according to the policy P3 i.e. 20% above and below the current load (4.6917*1.2 = 5.63 and 4.6917*0.8 = 3.7534). Note that if the upper threshold value is above the allocated bandwidth value then the upper threshold is set to the allocated bandwidth value. In order to avoid getting many threshold crossing events we defined a variable that sets the minimum time between two threshold crossings to 0.2 secs.

Table 6-1 DRsM Operation for the EF PHB

Time(s)	Thresh. Event	Offered Load	Req Bw	Bw Event	Alloc Bw	Lower Thresh	Curr Load	Upper Thresh
0.2001	AF1 UThr	2.5717	3.0861	Spare	3.3037	0.0000	2.5936	1.0000
0.2001	EF UThr	2.5717	3.0861	Spare	3.3037	2.0666	2.5833	3.1000
0.5685	AF1 UThr	2.9889	3.5867	Excess	3.3042	2.0666	2.9987	3.1000
2.0077	EF UThr	3.1784	3.6941	Excess	3.4204	2.4800	3.1000	3.4204
2.0574	AF1 LThr	4.2503	3.4002	Spare	4.6806	2.4800	3.4592	3.4204
2.2077	EF UThr	5.5993	6.7192	Excess	6.2213	3.7534	4.6917	5.6300
2.2577	AF1 LThr	5.7946	4.6357	Spare	6.3822	3.7534	6.2130	5.6300
2.4081	EF UThr	5.9851	7.1821	Excess	6.5914	4.9487	6.1859	6.5914

Table 6-2 DRsM Operation for the AF11 PHB

Time(s)	Thresh. Event	Offered Load	Req Bw	Bw Event	Alloc Bw	Lower Thresh	Curr Load	Upper Thresh
0.2001	AF1 UThr	5.2124	6.2549	Spare	6.6963	3.9933	4.9916	5.9899
0.2001	EF UThr	5.2124	6.2549	Spare	6.6963	3.9933	4.9916	5.9899
0.5685	AF1 UThr	6.0576	7.2691	Excess	6.6958	4.7920	5.9900	6.6958
2.0077	EF UThr	5.9216	7.1059	Excess	6.5796	4.7920	5.9324	6.5796
2.0574	AF1 LThr	4.8297	3.8638	Spare	5.3194	3.8307	4.7883	5.3194
2.2077	EF UThr	3.4007	4.0808	Excess	3.7787	3.7787	3.4681	3.7787
2.2577	AF1 LThr	3.2854	2.6283	Spare	3.6178	2.2151	2.7689	3.3226
2.4081	EF UThr	3.0949	3.7139	Excess	3.4086	2.2151	3.4250	3.3226

Figure 6-17 also illustrates the operation of the DRsM policies for both PHBs described above for the simple case scenario.

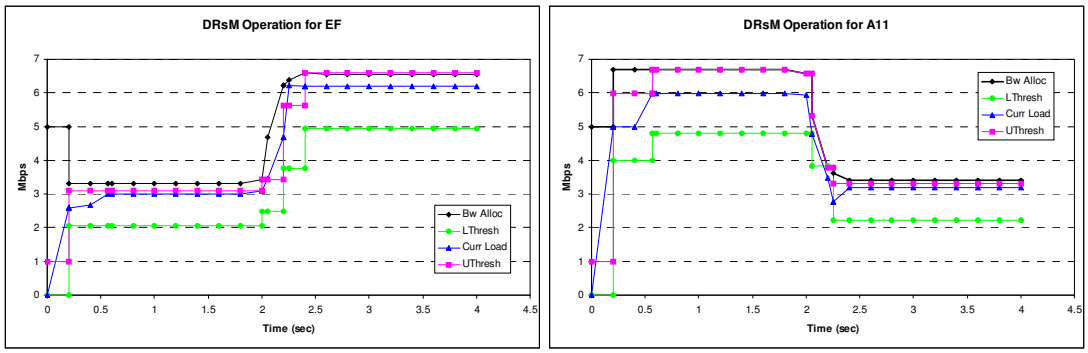


Figure 6-17 DRsM policies operation for EF and AF11 PHBs

In order to show the benefit of the DRsM policies, we performed a set of experiments where the same traffic was offered in two cases: a) 50% of the link capacity was allocated statically to each of the two queues, and b) the initial allocation of 50% to each queue was dynamically modified by the DRsM policies P3-10. Without loss of generality, we consider both Constant Bit Rate (CBR) traffic and Variable Bit Rate (VBR) traffic for this experiment with mean packet size equal to 500 bytes. When VBR sources are used, the offered load from each source refers to the average source

rate, and the standard deviation is set to 0.5Mbps. We investigate several situations by varying the total offered load (sum of EF and AF11 traffic) from 65% to 120% and compare the effect of the DRsM policies with the case of static bandwidth allocation for both the simple and complex scenario as shown in Figure 6-18.

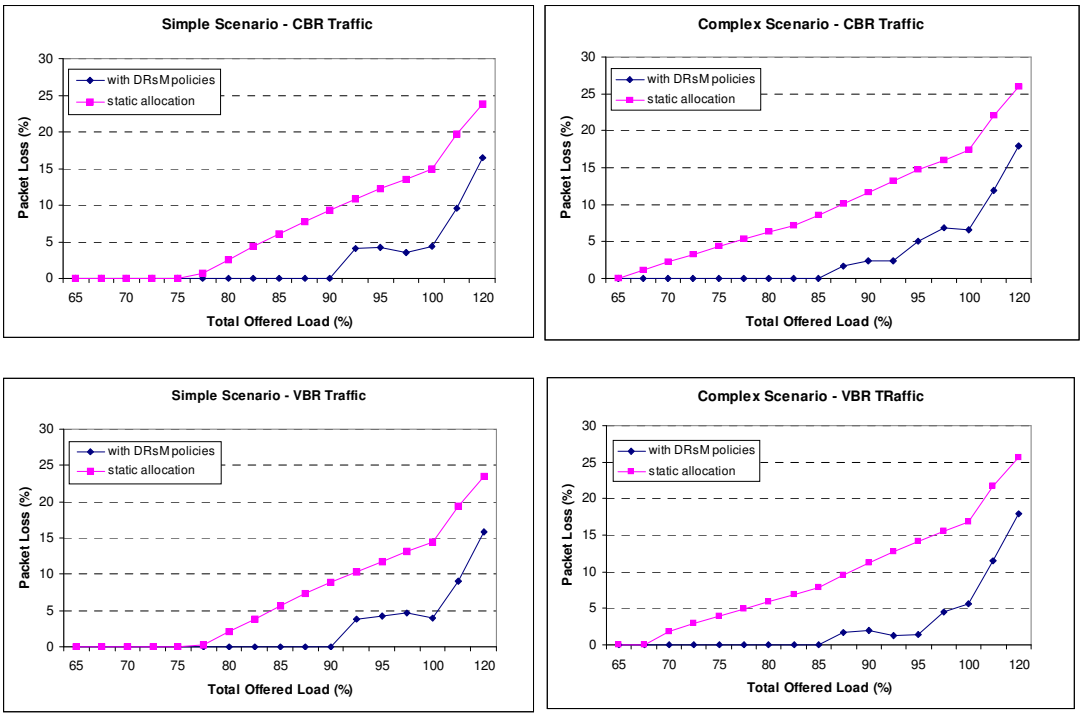


Figure 6-18 Comparison of static bandwidth allocation versus DRsM policies

It is obvious from the above figures that there is a substantial gain in terms of packet loss in scenarios i.e. simple and complex, using CBR and VBR traffic. Packet loss starts much later and becomes significant only very close to 100% offered load. This is obviously because the enforcement of the DRsM policies causes dynamic adaptation of the PHB allocated bandwidth to the current (actual) network load.

6.4 Summary and conclusions

In this chapter we presented the design and implementation of our prototype system that has been developed in order to provide a proof of concept validation of our work on QoS policies presented in the previous chapters. We focused on the implementation of the policy components that need to be deployed over the components of architecture presented in Chapter 4 and we presented specific examples of their enforcement on the Network Dimensioning and Dynamic Resource Management components.

We first described our implementation of the Policy Management Tool which provides a policy creation environment to the administrator and described the syntax of our proposed policy language. We described the functionality of the graphical user interface that provides a user-friendly interface to the administrator and gave specific examples of Network Dimensioning policy definitions.

We then described our implementation of the policy repository. We used the LDAP directory to realise it and we provided a CORBA interface identical to the LDAP protocol. Finally, we provided examples of how policies are stored as LDAP entries in the Directory showing their mapping from the information model based on PCIM and PCIMe presented in Chapter 5.

We finally presented the detailed design and implementation of the policy consumer which is considered as the most critical component since it is responsible for producing the code that dynamically enhances the management logic of our system. Examples in pseudo code are shown that were produced for realising the ND policy examples shown in the previous sections. Finally, we presented how we implemented our policy consumer over a simulated network in order to dynamically enforce Dynamic Resource Management policies based on actual network conditions. The effect of the enforcement of both the ND and the DRsM policies are also demonstrated.

The work presented in this chapter besides providing a proof of concept validation of the work presented in this thesis, it also provides a detailed insight as well as guidelines on how to implement policy-based systems, showing how policies are transformed from their definition until their enforcement. Such a description does not exist in the literature and constitutes a contribution in its own right. Most of the work presented in this chapter has been published in [Fleg03], [Trim02a] and [Trim02b].

Chapter 7

7 Conclusions and Future Work

In this final chapter, we bring together the work presented in the previous chapters of this thesis. Section 7.1 highlights the research contributions and discusses the importance of the main achievements. Section 7.2 identifies the areas in which work can be developed further, highlighting also some initial work which has been carried out towards the end of the completion of this thesis.

7.1 Contributions and Main Achievements of this Thesis

The main contribution of this thesis was the investigation of applying Policy-based Management in the context of QoS Management of IP DiffServ Networks. By using policies as a means for building extensible hierarchical management systems, we proposed a novel Policy-based QoS management architecture and specified the relevant policies that can drive its behaviour dynamically, providing a holistic approach to the area of policies for QoS Management.

The main achievements of this thesis are of architectural nature and have been backed up by design, implementation and experimental results for demonstration purposes proving the feasibility of the proposed work. The research contribution of this thesis were summarised at the end of Chapters 3, 4, 5 and 6. We re-iterate through the main achievements below.

Achievements related to our view on using policies to build extensible hierarchical management systems:

- We presented our view on policies as a means to build extensible, flexible and programmable systems. The relationship with mobile code paradigms and approaches used in active networks was also extensively discussed and justified. The above characteristics of Policy-based management have never been explicitly presented in the literature and as such this constitutes a research contribution in its own right.
- We proposed generic guidelines for designing policy-based systems and explored further the issue on where to draw the line between static management intelligence and logic dynamically introduced in the system through policies.

- We studied the coexistence of policies with hierarchical management systems and proposed a generic framework. The issue of high-level policy refinement into policies enforced in every layer of the management hierarchy was introduced and the cases when this methodology is applicable were identified.
- We proposed a generic framework for dynamically creating management services in the system through policies. The translation of policies into interpreted code which represents the logic that combines components modelling the hardwired functionality of the system enables the creation of services while the system is up and running.

The contributions related to the policy-based QoS Management Architecture are the following:

- The proposed architecture provides a holistic approach to intra-domain QoS, including both service management and traffic engineering functionality. All the sub-systems of the architecture were presented and decomposed into functional blocks providing detailed descriptions of their functionality and interactions.
- We adopted a two-level hierarchical approach to both Service Management and Traffic Engineering by incorporating dynamic service layer and traffic engineering functions in addition to the corresponding offline components. This approach is not monolithic, as pure centralized schemes in the literature, while at the same time it harnesses the dynamics of state/load-dependent schemes on the basis of guidelines produced with a longer-term vision.
- We showed how policies can be applied to the proposed architecture following our methodology for applying policies to hierarchical distributed management systems
- We presented a generic categorisation of QoS policies applied to our architecture depending on the "position" of the components they influence. The above classification differs a lot from the QoS policies identified by other researchers and the IETF which are limited to policies configuring the edge routers of a DiffServ/IntServ network while our work provides a holistic view of QoS policies starting from SLS Management policies, network-wide resource management policies down to router management policies.

The main achievements related to the definition of QoS policies are the following:

- Following our proposed generic classification of QoS policies, we presented SLS management and Traffic Engineering policies driving the offline as well as the dynamic management components of both sub-systems.
- We defined SLS-subscription and invocation management policies that influence the admission control decisions at both levels and they are mostly triggered by conditions

related to characteristics of SLS subscriptions and invocations, enabling differential treatment on different types of SLSs. While most of the work on SLA related policies focused on how to derive low-level device configuration in order to realise SLAs, our work defined policies that guide the behaviour of the management system on how to handle SLSs based on the business objectives of the ISP.

- We defined Traffic Engineering policies that influence the resource allocation and the routing decisions of the different supported QoS classes at both levels i.e. offline and dynamic. While most of the related work QoS policies focused on how to configure the nodes of a DiffServ network in a static manner, our work proposed network-wide resource management policies as well as policies that adapt the configuration of the network based on the actual network status.
- We presented a formal representation of the above policies using the object-oriented Policy Core Information Model (PCIM) and its extensions (PCIMe) as defined jointly by the IETF and DMTF standardisation bodies. For this reason, we extended the core classes provided by the aforementioned models by defining specific classes and their properties needed to represent the SLS management and Traffic Engineering policies.
- The policies presented in this thesis provide an important contribution in the area of QoS policies since they provide a holistic approach of the management functionality which can be implemented using policies, in contrast to the simple configuration policies presented in the literature.
- Using as a case study the proposed QoS policies, we validated the methodology for applying policies to hierarchical management systems. We focused on the hierarchical relationship of the ND and DRsM components and discussed the situation where an ND policy could possibly be refined into a DRsM policy by giving a specific example.
- We showed how we can combine the static functionality of the DRsM component with the relevant policies in order to dynamically create a DRsM service in our system following the proposed framework for management service creation using policies.

Finally, we presented the design and implementation of our prototype system that has been developed to provide a proof of concept validation of the proposed work on policy-based QoS Management. This design and implementation besides providing a proof of concept validation of the work presented in this thesis, it also provided a detailed insight as well as guidelines on how to implement policy-based systems, showing how policies are transformed from their definition until their enforcement. Such a clear and detailed demonstration of all levels of policy functionality has never been presented before in the literature.

7.2 Directions for Future Work

There are various ways that the work presented in this thesis can be taken further. We summarise potential future directions below.

As discussed in Chapter 3, in policy-based systems, management intelligence does not follow the rigid analysis, design, implementation, testing and deployment cycle, and as such, conflicts may be the norm rather than the exception. Conflict analysis and detection is required both statically, at policy introduction and deployment time, and also dynamically, at run time. Initial work has been proposed in the area of conflict analysis [Band03] that performs a priori analysis of policy specifications for the generic conflict types presented in the literature. While this work proposes a promising methodology to tackle the problem of conflict analysis in a generic fashion, it is not sufficient enough to provide a complete solution to the problem without addressing the needs of an application-specific domain such as the QoS Management of IP Networks. Since in the context of this thesis we have shown how to drive a QoS Management system through policies and identified precisely the relevant QoS policies, the next step is to identify conflicts that may arise among the policies defined in this thesis and provide resolution methods.

The above proposed future work is a difficult task but essential for the deployment of the proposed work of this thesis. In order to achieve this task, a good understanding of the impact of the QoS policies on the behaviour of the components is required for identifying the conflicting situations. Although conflicts may be identified between policies that apply to a single architectural component of the architecture defined in this thesis, conflicts may arise between policies that apply to different components e.g. a Network Dimensioning policy may conflict with a Dynamic Resource Management policy. This means that the effect of policies applied to the whole architecture should be studied and their relationship should be identified with respect to potential conflicts. Initial work has already been published in [Char05] where the author of this thesis was one of the major contributors. In this initial work, we presented a conflict analysis methodology for the network dimensioning policies, presenting a taxonomy of the potential conflicts that may arise. The study of the scalability of the proposed policy-driven system with respect to number of introduced policy rules is also a very important issue that has not been covered in this thesis. Scalability can be measured both in terms of conflicts that may arise as well as in terms of the performance of the policy-based dynamic components since the latter have to react immediately to unpredicted network conditions based on actual traffic measurements.

Another important area for future direction is the investigation of policy refinement techniques applied to the QoS policies defined in this thesis. Although these policies cover all areas of QoS management i.e. Service Management and Traffic Engineering, they are all low-level policies that can be directly translated and enforced on the components of our architecture. Policy refinement

is the process of transforming a high-level, abstract policy specification into a series of low-level, concrete ones. This derivation process is usually performed manually with no means of verifying that the policies written are supported by the policy targets and actually achieve the desired high-level goals. The proposed future direction is to extend existing approaches and apply them to the domain of QoS management, enabling the specification of high-level QoS related goals that should be as much as possible automatically refined to low-level policies as the ones presented in this thesis. Initial work towards this direction was presented in [Band05] where the author of this thesis was one of the major contributors. The approach makes use of goal elaboration and abductive reasoning to derive strategies that will achieve a given high-level goal. By combining these strategies with the appropriate events and constraints, we provided an initial approach and showed how high-level QoS policies can be refined, what tool support can be provided for the refinement process and demonstrated how application-specific (QoS related) policy refinement patterns are identified and specified.

Finally, the last potential future direction of this work is to extend it into the area of inter-domain QoS. The architecture and the policies presented in this thesis provide a holistic approach for achieving QoS within a single administrative domain. The proposed architecture needs to be extended with a set of functions in the management, control and data planes required for a network provider to provide end-to-end QoS-based IP connectivity services. The ability to deliver inter-domain QoS requires different ISPs to negotiate service contracts with each other; and to engineer their networks to provide the required level of performance. The author of this thesis has contributed to the specification of such an architecture for achieving end-to-end QoS in the Internet in the context of his involvement in the MESCAL EU IST project [MESCAL], which has been published in [Howa05a] and [Howa05b]. In the same manner we defined policies for the intra-domain QoS Management architecture, policies driving the inter-domain components should be defined and their relationship with the intra-domain policies should be studied. Applying policies related to inter-domain QoS is a very challenging area since the latter involves service management and traffic engineering decisions which are very much dependent on the business goals and relations of the provider with its peer providers. A very interesting topic in this direction is that of conflict analysis that involves conflicting policies between peering Network Providers.

Bibliography

[Alpers95] B. Alpers, H. Plansky, *Concepts and application of Policy-Based Management*, Proceedings of the fourth International Symposium on Integrated Network Management (ISINM'95), Santa Barbara, California, USA, pp. 57-68, May 1-5, 1995.

[Auki00] P. Aukia, M. Kodialam, P.V.N. Koppol, T.V. Lakshman, H. Sarin, and B. Suter, *RATES: A Server for MPLS Traffic Engineering*, IEEE Network Magazine, March/April 2000.

[Band03] A. K. Bandara, E. C. Lupu, A. Russo, *Using Event Calculus to Formalise Policy Specification and Analysis*, Proceedings 4th IEEE Workshop on Policies for Distributed Systems and Networks (Policy 2003) Lake Como, Italy, June 2003

[Band05] A. Bandara, E. Lupu, A. Russo, N. Dulay, M. Sloman, P. Flegkas, M. Charalambides, G. Pavlou, *Policy Refinement for DiffServ Quality of Service Management*, to appear in the proceedings of the IEEE/IFIP Integrated Management Symposium (IM'2005), Nice, France, IEEE, May 2005

[BCIT] http://www.bcit.ca/appliedresearch/gait/projects/bandwidth.shtml

[Berts99] D. Bertsekas, *Nonlinear Programming*, (2nd ed.) Athena Scientific, 1999

[BISDN] ITU-T Rec. I.320-321, *Broadband ISDN Reference Model*

[Brun01] M. Brunner and J. Quittek, *MPLS Management using Policies*, proceedings of the IEEE/IFIP International Symposium on Intergrated Network Management (IM2001), Seattle, WA, USA, , pp. 515-528, 14-18 May 2001

[CANE] CANEs: Composable Active Network Elements, http://www.cc.gatech.edu/projects/canes/

[Char05] M. Charalambides, P. Flegkas, G. Pavlou, A. Bandara, E. Lupu, A. Russo, N. Dulay, M. Sloman, *Policy Conflict Analysis for Quality of Service Management*, to appear in the proceedings of the IEEE Workshop on Policy for Distributed Systems and Networks (Policy 2005), Stockholm, Sweden, IEEE, June 2005

[CIM] DMTF, Common Information Model (CIM) Version 2.9 Specification

[CORBA] Open Management Group (OMG), *The Common Object Request Broker: Architecture and Specification (CORBA)*, available online at: http://www.omg.org/technology/documents

[Corr01] A. Corradi, N. Dulay, R. Montanari, C. Stefanelli, *Policy-Driven Management of Agent Systems*, proceedings of the Policy 2001: International Workshop on Policies for Distributed Systems and Networks, Bristol, UK, Springer-Verlag, pp. 137-152, January 2001

[D1.2] P. Trimintzios, et al., *D1.2: Protocol and Algorithm Specification*, TEQUILA deliverable, available from http://www.ist-tequila.org

[Dami01] N. Damianou, Dulay N, Lupu, E, Sloman M, "The Ponder Policy Specification Language", *Proc. Policy 2001:Workshop on Policies for Distributed Systems and Networks,* Bristol, UK, Jan. 2001, Springer-Verlag LNCS 1995

[Dami02] N. Damianou, *A Policy Framework for Management of Distributed Systems,* PhD Thesis, March 2002

[diffserv-wg] IETF DiffServ Working Group, *http://www.ietf.org/html.charters/OLD/diffserv-charter.html*

[DMTF] Distributed Management Task Force, http://www.dmtf.org

[Feld00] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, and J. Rexford. *NetScope: Traffic Engineering for IP Networks*, IEEE Network, March/April 2000

[Feld01] A. Feldmann and J. Rexford, *IP Network Configuration for Intra-domain Traffic Engineering*, IEEE Network Magazine, vol. 15, no. 5, pp. 46-57, September/October 2001

[Fern01] M. Fernandez et al., *QoS Provisioning across a DiffServ Domain using Policy-based Management*, Proceedings of IEEE Globecom 2001, San Antonio, Texas, pp. 2220-2224, June 2001

[Fleg01] P. Flegkas, P.Trimintzios, G. Pavlou, I. Andrikopoulos, C. F. Cavalcanti, *On Policy-based Extensible Hierarchical Network Management in QoS-enabled IP Networks*, Proceedings of the IEEE Workshop on Policies for Distributed Systems and Networks (Policy '01), Bristol, UK, M. Sloman, J. Lobo, E. Lupu, eds., pp. 230-246, Springer-Verlag, January 2001

[Fleg02] P. Flegkas, P. Trimintzios, G. Pavlou, *A Policy-based Quality of Service Management System for IP Differentiated Services Networks*, IEEE Network,

special issue on Policy Based Networking, Vol. 16, No. 2, pp. 50-56, IEEE, March/April 2002

[Fleg03] P. Flegkas, P. Trimintzios, G. Pavlou, A. Liotta, *Design and Implementation of a Policy-based Resource Management Architecture*, Proceedings of IEEE/IFIP Integrated Management Symposium (IM'2003), Colorado Springs, Colorado, USA, Kluwer, pp. 215-229, March 2003

[Floy93] S. Floyd, and V. Jacobson, *Random Early Detection gateways for Congestion Avoidance*, IEEE/ACM Transactions on Networking, Vol.1, No.4, pp. 397-413 August 1993

[Fugg98] A. Fuggetta, G. P. Picco, G. Vigna, *Understanding Code Mobility*, IEEE Transactions on Software Engineering, vol. 24, no. 5, pp. 342-361, 1998

[Georg99] P. Georgatsos, D. Makris, D. Griffin, G. Pavlou, S. Sartzetakis, Y. T'Joens, D. Ranc, *Technology Interoperation in ATM Networks: the REFORM System*, IEEE Communications, Vol. 37, No. 5, pp. 112-118, IEEE, May 1999

[God01] D. Goderis (ed.). *Service Level Specification Semantics and Parameters*. Internet draft, draft-tequila-sls-01.txt , December 2001. work in progress, available at: www.ist-tequila.org/sls

[Hick98] M. Hicks et al., *PLAN: A Programming Language for Active Networks*, Proccedings of the ICFP'98, 1998.

[Hoo99] G. Hoo, W. Johnston, I. Foster, A. Roy, *QoS as Middleware: Bandwidth Reservation System*, Proceedings of the 8th IEEE Symposium on High Performance Distributed Computing, 1999

[Howa05a] M. P. Howarth, P. Flegkas, G. Pavlou, N. Wang, P. Trimintzios, H. Asgari, D. Griffin, J. Griem, M. Boucadair, P. Morand, P. Georgatsos, *Provisioning for Inter-domain Quality of Service: the MESCAL Approach*, to appear in IEEE Communications, 2005

[Howa05b] M. P. Howarth, M. Boucadair, P. Flegkas, N. Wang, G. Pavlou, P. Morand, T. Coadic, D. Griffin, A. Asgari, P. Georgatsos, *End-to-end quality of service provisioning through Inter-provider traffic engineering*, to appear in Computer Communications, Special Issue on End-to-End QoS, 2005

[Kim03] T. K. Kim, D. Y. Lee, O. H. Byeon and T.M. Chung, *A Policy Propagation Model Using Mobile Agents in Large-Scale Distributed Network Environments*, Proceedings of the First International Conference on Service Oriented Computing (ICSOC2003), pp. 514 – 526, Italy, December 2003

[Koch95] T. Koch, B. Krämer , G. Rohde, *On a rule based management architecture*, Proceedings of the Second International Workshop on Services in Distributed and Networked Environments, IEEE Computer Society Press, pp. 68—75, 1995.

[Lams01] A. van Lamsweerde, *Goal-Oriented Requirements Engineering: A Guided Tour*, proceedings of the 5th IEEE International Symposium on Requirements Engineering, Toronto, pp. 249-263, August 2001

[Lymb03] L. Lymberopoulos, E. Lupu and M. Sloman, *An Adaptive Policy Based Framework for Network Services Management*, Plenum Press Journal of Network and Systems Management, Special Issue on Policy Based Management, Vol 11, No. 3, p277-303, September 2003

[Lymb04] L. Lymberopoulos, E. Lupu and M. Sloman, *Ponder Policy Implementation and Validation in a CIM and Differentiated Services Framework,* Proceedings of the 9th IEEE/IFIP Network Operations and Management Symposium (NOMS 2004), Seoul, Korea, 19-23 April 2004

[M3010] ITU-T Rec. M.3010, *Principles for a Telecommunications Management Network (TMN),* Study Group IV, 1996

[Marr96] D. Marriott, M. Sloman, *Implementation of a Management Agent for Interpreting Obligation Policy*, Proceedings of the seventh IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM'96), L'Aquila, Italy, October 28-30, 1996

[Marr97] Damian Marriott, *Policy Service for Distributed Systems*, PhD thesis, October 1997

[Mart02] M. Martinez, M. Brunner, J. Quittek, F. Strauß, J. Schönwälder, S. Mertens and T. Klie, *Using the Script MIB for Policy-based Configuration Management*., proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS2002), Florence, Italy, pp 187-202, April 2002

[MESCAL] EU IST MESCAL, *Management of End-to-end Quality of Service Across the Internet at Large*, information available at: http://www.mescal.org

[Moff93] J. Moffett, M. Sloman, *Policy Hierarchies for Distributed Systems Management*, IEEE Journal on Selected Areas in Communications, Vol. 11, No. 9, pp. 1404-1414, December 1993

[Myko03] E. Mykoniati, C. Charalampous, P. Georgatsos, T. Damilatis, D. Goderis, P. Trimintzios, G. Pavlou, D. Griffin, *Admission Control for Providing QoS in IP*

DiffServ Networks: the TEQUILA Approach, IEEE Communications, special issue on QoS Advances: the European IST Premium IP Projects, Vol. 41, No. 1, pp. 38-44, IEEE, January 2003

[Neil99] R. Neilson, J. Wheeler, F. Reichmeyer, S. Haresm, *A Discussion of Bandwidth Broker Requirements for Internet2 Qbone Deployment*, August 1999.

[Netpolicy] Allot Communications, *NetPolicy*, Policy Based Management System, product documentation, available from http://www.allot.com

[NS] The Network Simulator – ns-2, http://www.isi.edu/nsnam/ns/

[openldap] Open Source Implementation of LDAP, http://www.openldap.org

[Pana04] M. Pana, et al., *Policy Core Extension Lightweight Directory Access Protocol Schema*, <draft-reyes-policy-core-ext-schema-07.txt>, internet draft, IETF, October 2004

[Pavl04] G. Pavlou, P. Flegkas, S. Gouveris, A. Liotta, *On Management Technologies and the Potential of Web Services*, IEEE Communications, special issue on XML-based Management of Networks and Services, Vol. 42, No. 7, pp. 58-66, IEEE, July 2004

[Pavl94] G. Pavlou, The Telecommunications Management Network (TMN), Tutorial presented in the 2nd International Conference on Intelligence in Services and Networks (IS&N'94), Aachen, Germany, September 1994

[Pavl96] G. Pavlou, D. Griffin, *TMN Architecture Issues*, in *Integrated Communications Management of Broadband Networks*, , pp. 49-71, Crete, University Press, 1996

[Pavl98] G. Pavlou, D. Griffin, *An Evolutionary Approach Towards the Future Integration of IN and TMN*, Interoperable Communications Networks Journal, Special Issue on Service Engineering, Vol. 1, pp. 1-16, Baltzer Science Publishers, 1998

[policy-wg] IETF Policy Working Group, *http://www.ietf.org/html.charters/OLD/policy-charter.html*

[PolicyXpert] HP Policy Expert, http://www.openview.hp.com/products/pxpert/index.html

[Ponn02] A. Ponnappan, L. Yang, R. Pillai, *A Policy Based QoS Management System for the IntServ/DiffServ Based Internet*, Proceedings of the Third International Workshop on Policies for Distributed Systems and Networks (POLICY '02), Monterey, California, June 2002

[Putt95] P. Putter, J. Bishop, J. Roos, *Towards policy driven systems management*, Proceedings of the fourth International Symposium on Integrated Network Management (ISINM'95), Santa Barbara, California, USA, pp. 69-80, May 1-5, 1995.

[Q1200] ITU-T Rec. Q1200 series, General Series Intelligent Networks Recommendations Structure, 1992

[QBONE] QBONE, http://www.internet2.edu/qos/wg/papers/qbArch/

[QPM] Cisco Systems Inc., *COPS QoS Policy Manager*, available from http://www.cisco.com

[rap-wg] IETF Policy Working Group, *http://www.ietf.org/html.charters/rap-charter.html*

[RFC1633] R. Braden, D. Clark, S. Shenker, *Integrated Services in the Internet Architecture: an Overview*, RFC1633, IETF, July 1994

[RFC2209] R. Braden, L. Zhang, *Resource ReSerVation Protocol (RSVP) - Version 1 Message Processing Rules*, RFC 2209, IETF, September 1997

[RFC2211] J. Wroclawski, *Specification of the Controlled-Load Network Element Service*, RFC2211, IETF, September 1997

[RFC2212] S. Shenker, C. Partridge, R. Guerin, *Specification of Guaranteed Quality of Service*, RFC2212, IETF, September 1997

[RFC2251] M. Wahl, T. Howes, S. Kille, *Lightweight Directory Access Protocol (v3)*, RFC 2251, December 1997

[RFC2474] K. Nichols, et al., *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*, RFC2474, December 1998

[RFC2475] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang and W. Weiss, *An Architecture for Differentiated Services*, RFC2475, IETF, December 1998

[RFC2592] D. Levi, and J. Schoenwaelder, *Definitions of Managed Objects for the Delegation of Management Scripts*, RFC 2592, IETF, May 1999

[RFC2597] J. Heinanen, F. Baker, W. Weiss and J. Wroclawski, *Assured Forwarding PHB Group*, RFC2597, June 1999

[RFC2598] V. Jacobson, K. Nichols, K. Poduri, *An Expedited Forwarding PHB*, RFC2598, IETF, June 1999

[RFC2638] K. Nichols, V. Jacobson, and L .Zhang, *A Two-bit Differentiated Services Architecture for the Internet*, RFC 2638, IETF, July 1999

[RFC2748] D. Durham et al., *The COPS (Common Open Policy Service) Protocol*, RFC 2748, IETF, January 2000

[RFC2749] S. Herzog et al., *COPS Usage for RSVP*, RFC 2749, IETF, January 2000

[RFC2753] R. Yavatkar, D. Pendarakis, R. Guerin, *A Framework for Policy Based Admission Control*, Informational RFC 2753, IETF, January 2000

[RFC3031] E. Rosen, A. Viswanathan, R. Callon, *Multiprotocol Label Switching Architecture*, RFC3031, IETF, January 2001

[RFC3060] B. Moore, E. Ellesson, J. Strassner and A. Westerinen (2001), Policy Core Information Model -- Version 1 Specification, RFC 3060, IETF, February 2001

[RFC3084] K. Chan et al., *COPS Usage for Policy Provisioning (COPS-PR)*, RFC 3084, IETF, March 2001

[RFC3086] K. Nichols and B. Carpenter. Definition of Differentiated Services Per Domain Behaviors and Rules for their Specification. RFC 3086, IETF, April 2001

[RFC3272] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, and X. Xiao. Overview and principles of Internet Traffic Engineering. RFC 3272, IETF, May 2002

[RFC3317] K. Chan, et al., *Differentiated Services Quality of Service Policy Information Base,* RFC 3317, IETF, March 2003

[RFC3318] R. Sahita, et al., *Framework Policy Information Base*, RFC 3318, IETF, March 2003

[RFC3460] B. Moore et al., *Policy Core Information Model (PCIM) extensions*, RFC 3460, IETF, January 2003

[RFC3564] F. Le Faucheur et al., *Requirements for Support of Differentiated Services-aware MPLS Traffic Engineering*, RFC 3564, IETF, July 2003

[RFC3644] Y. Snir, Y. Ramberg, J. Strassner, R. Cohen, B. Moore, *Policy Quality of Service (QoS) Information Model*, RFC 3644, IETF, January 2003

[RFC3703] J. Strassner, B. Moore, R. Moats, E. Ellesson, *Policy Core Lightweight Directory Access Protocol (LDAP) Schema*, RFC3703, IETF, February 2004

[SableCC] E. Gagnon. SableCC, *An Object-Oriented Compiler Framework, Master of Science*, School of Computer Science, McGill University, Montreal

[Schw94] H. Schwingel-Horner, G. Bonn, *IDSM Authorisation Policy Specification and Enforcement in a Hierarchical Management Environment*, Proceedings of the

fifth IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM'94), Toulouse, France, October 10-12, 1994.

[Silv98] S. Da Silva, D. Florissi, and Y. Yemini, *NetScript: A Language-Based Approach to Active Networks*, Technical Report, Computer Science Dept., Columbia University, January 1998

[Slom89] M. Sloman, J. Moffet, *Domain Management for Distributed Systems*, Integrated Network Management I, B. Meandzija and J. Westcott, eds., pp. 505-516, North Holland, 1989.

[Slom94] M. Sloman, *Policy Driven Management For Distributed Systems*, Journal of Network and Systems Management, Vol. 2, No. 4, pp. 333-360, Plenum Publishing, December 1994.

[Slom99] M. Sloman, E. Lupu, *Policy Specification for Programmable Networks*, Proc. of the 1st International Conference on Active Networks, Berlin, Germany, ed. S. Covaci, Springer Verlag, June 1999.

[SNMP] J.Case, M.Fedor, M.Schoffstall, J.Davin, *A Simple Network Management Protocol (SNMP)*, IETF RFC 1157, 1990

[Stev99] M. Stevens et al., *Policy Framework*, Internet Draft, draft-ietf-policy-framework-00.txt, September 1999

[Ston01] G. Stone, B. Lundy and G. Xie, Network Policy Languages: A Survey and A New Approach, IEEE Network Magazine, Vol. 15, No 1, pp. 10-21, January 2001

[Tenn97] D. Tennenhouse, J. Smith, W. Sincoskie, D. Wetherall, G. Minden, *A Survey of Active Network Research*, IEEE Communications Magazine, vol. 35, no. 1, pp. 80-86, January 1997

[TEQUILA] EU IST TEQUILA, *Traffic Engineering for Quality of Service in the Internet, at Large Scale*, project number: IST–1999–11253, information available at: http://www.ist-tequila.org

[Terz99] A. Terzis, J. Ogawa, S. Tsui, L. Wang, L. Zhang. *A Prototype Implementation of the Two-Tier Architecture for Differentiated Services*, appeared in RTAS99 Vancouver, Canada, 1999

[Trim01] P. Trimintzios, I. Andrikopoulos, G. Pavlou, P. Flegkas, D. Griffin, P. Georgatsos, D. Goderis, Y. T'Joens, L. Georgiadis, C. Jacquenet, R. Egan, *A Management and Control Architecture for Providing IP Differentiated Services*

in MPLS-based Networks, IEEE Communications, special issue in IP-Oriented Operations and Management, Vol. 39, No. 5, pp. 80-88, IEEE, May 2001

[Trim02a] P. Trimintzios, P.Flegkas, G. Pavlou, L. Georgiades, D. Griffin, *Policy-based Network Dimensioning for IP Differentiated Services Networks*, Proceedings of the IEEE Workshop on IP Operations and Management (IPOM 2002), Dallas, Texas, USA, pp. 171-176, October 2002

[Trim02b] P. Trimintzios, P.Flegkas, G. Pavlou, *Policy-driven Traffic Engineering for Intra-domain Quality of Service Provisioning*, Proceedings of Quality of Service for future Internet Services (QofIS'02), Zurich, Switzerland, Springer, pp. 179-193, October 2002

[Trim03] P. Trimintzios, G. Pavlou, P. Flegkas, P. Georgatsos, A. Asgari, E. Mykoniati, *Service-driven Traffic Engineering for Intra-domain Quality of Service Management*, IEEE Network, special issue on Network Management of Multiservice, Multimedia, IP-based Networks, Vol. 17, No. 3, pp. 29-36, IEEE, May/June 2003

[Trim03b] P. Trimintzios, T. Bauge, G. Pavlou, P. Flegkas, R. Egan, *Quality of Service Provisioning through Traffic Engineering with Applicability to IP-based Production Networks*, Computer Communications, special issue on Performance Evaluation of IP Networks and Services, Vol. 26, No. 8, pp. 845-860, Elsevier Science Publishers, May 2003

[Verm01a] D. Verma, *Policy-Based Networking, Architecture and Algorithms*, New Riders Publishing, 2001

[Verm01b] D. Verma, M. Beigi and R. Jennings, *Policy Based SLA Management in Enterprise Networks*, proceedings of the Policy 2001: International Workshop on Policies for Distributed Systems and Networks, Bristol, UK, Springer-Verlag, pp. 137-152, January 2001

[Wang96] Z. Wang, and J. Crowcroft, *Quality of Service Routing for Supporting Multimedia Applications*, IEEE Journal of Selected Areas in Communications, vol. 14, no. 7, pp. 1228-1234, September 1996

[Wies95] R. Wies, *Policies in Integrated Network and Systems Management: Methodologies for the Definition, Transformation and Application of Management Policie*, PhD thesis, June 1995.

[X500] ITU-T Rec. X.500, Information Technology - Open Systems Interconnection, *The Directory: Overview of Concepts, Models and Service,* 1993

[X700] ITU-T Rec. X.700, Information Technology - Open Systems Interconnection, *OSI Management Framework*, 1992

[Yang02] K. Yang., A. Galis ,T. Mota, S. Gouveris, Automated Management of IP Networks Through Policy and Mobile Agents, proceedings of the IEEE/ACM International Workshop on Mobile Agents for Telecommunication Applications (MATA'02), Barcelona, Spain, Springer, pp. 249-258, October 2002

[Yemi91] Y. Yemini, G. Goldszmidt, S. Yemini, *Network Management by Delegation*, in Integrated Network Management II, Krishnan, Zimmer, eds., pp. 95-107, Elsevier, 1991

[Zhan00] Z.L. Zhang, et al. *Decoupling QoS Control from Core Routers: A Novel Bandwidth Broker Architecture for Scalable Support of Guaranteed Services*, Proceedings of ACM SIGCOMM2000, Sweden, August 2000

Appendix A - Policy Language Specification

```
/************************************************/
/*    Policy Language Specification File        */
/*          Paris Flegkas                       */
/*      University of Surrey                    */
/************************************************/

Package policy;

Helpers

   letter_small         = ['a'..'z'];
   letter_caps          = ['A'..'Z'];
   digit                = ['0'..'9'];
   letter               = letter_small|letter_caps|'_';
   letter_or_digit      = letter|digit|'/';

   date_helper          = '1'|'0';
   time_hour            = ['0'..'3'];
   time_minute          = ['0'..'5'];
   time1                = '0' digit time_minute digit;
   time2                = '1' digit time_minute digit;
   time3                = '2' time_hour time_minute digit;

   equal_helper         = '= ';



Tokens


add         ='Add'|'ADD'|'add';
retrieve    ='Retrieve'|'RETRIEVE'|'retrieve';
delete      ='Delete'|'delete'|'DELETE';
blank       = (' ' | 13 | 10)+;


 /************* Group - Rule  Name ****************/

      rulegroup   = letter letter_or_digit*;

 /***************** Conditions *********************/


   if             = 'if '|'IF '|'If ';
   and            = 'and '|'And '|'AND ';
   or             = 'or '|'OR '|'Or ';

/*part of the ND policy conditions*/
   ordered_agg    = 'Ordered Aggregate ';
   ord_agg_value  = 'EF '|'AF1 '|'AF2 '|'AF3'|'BE ';
```

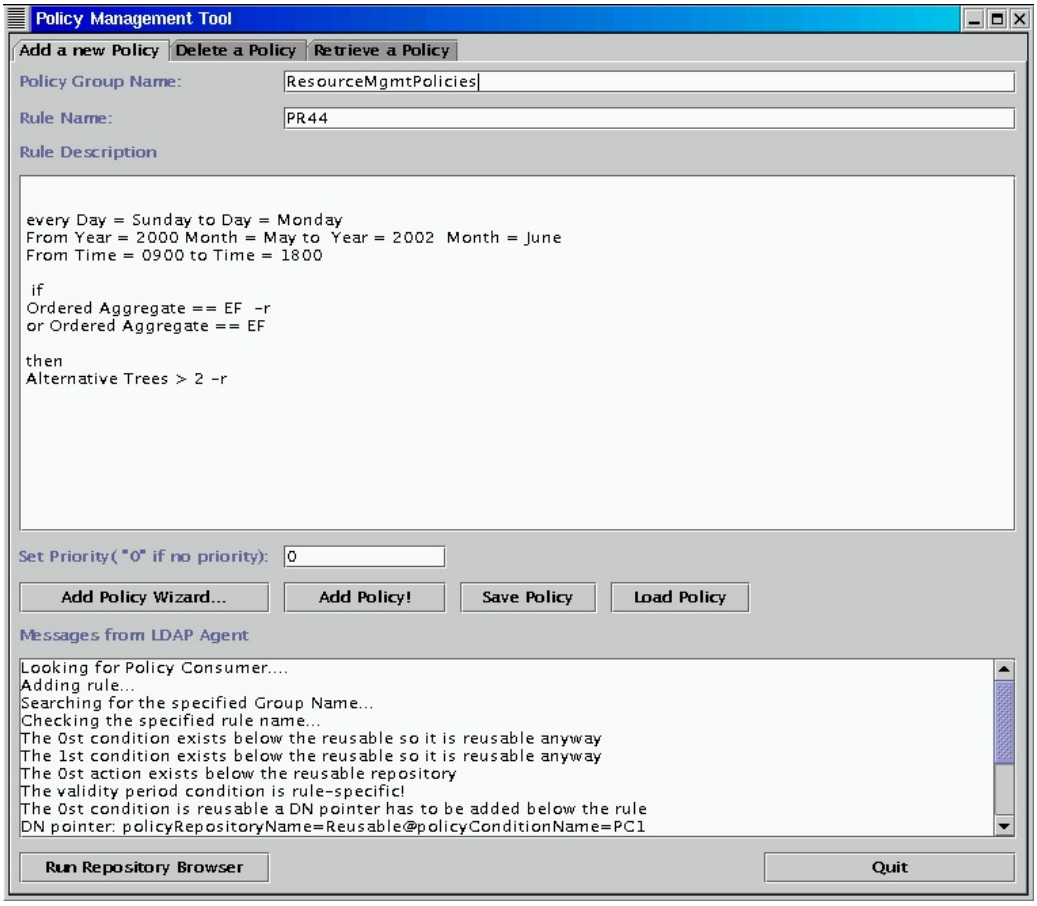
```
   spare_bw             = 'Spare Bw ';
   spareover_bw_value   ='True '|'true '|'TRUE '|'false '|'False
                        '|'FALSE ';
   over_bw              = 'Overprovisioned Bw ';
   ……

   left_par       = '(';
   right_par      =')';
   reusable       = '-r';
   not            = 'not ';
   time_and       ='&';

  /***************** Actions **********************/

   then                 = 'then '|'Then '|'THEN ';

/*part of the ND policy actions*/

   alternative_trees    = 'Alternative Trees';
   link_cost            = 'Link Cost';

 /*********   Time Period Condition ***************/

   time_or_year         =time1|time2|time3;
   label_time           ='Time '|'time '|'TIME ';

   from     ='from '|'From '|'FROM ';
   to       ='to '|'To '|'TO ';
   every    ='Every '|'every '|'EVERY ';
   day      ='Day '|'day '|'DAY ';



   date         ='Date '|'DATE '|'date ';
   month        ='month '|'Month '|'MONTH ';
   year         ='year '|'Year '|'YEAR ';



   dayname       ='Monday '|'Friday '|'Tuesday '|'Wednesday '|'Thursday
'|'Sunday '|'Saturday ';
   monthname     = 'January '|'February '|'March '|'April '|'May '|'June
'|'July '|'August '|'September '|'October '|'November '
                 |'December ';


/*****************Special Characters ***************/

number          = digit*; /*digit|digit digit|digit digit digit|digit
digit digit digit;*/

dequals         ='==';
equals          = equal_helper;

slash           ='/';
action_symb     = '< '|'> ';
priority        = digit;



/*****************Ignored Tokens*******************/
```

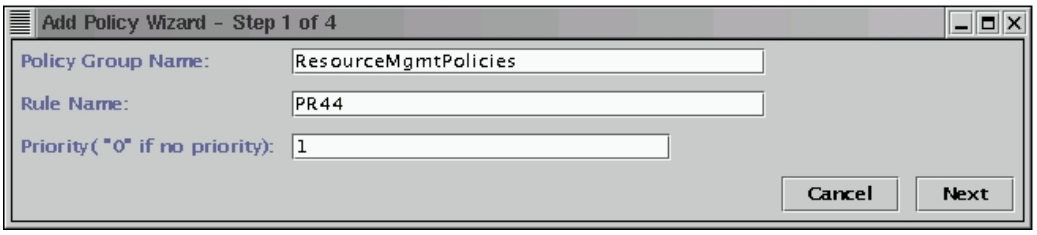
```
 Ignored Tokens
 blank;


/******************Productions**********************/


Productions

  operation =

  {add}           addpolicy|
  {retrieve}      retrieve [group]:rulegroup [rule]:rulegroup|
  {retrieverule}  retrieve rulegroup|
  {delete}        delete[group]:rulegroup [rule]:rulegroup|
  {deleterule}    delete rulegroup;


  addpolicy =

       {cnf}      add[group1]:rulegroup [rule1]:rulegroup timeperiod? if
cnf then action number?|
       {dnf}      add[group]:rulegroup [rule]:rulegroup timeperiod? if
dnf then action number?|
       {only_or}  add[group]:rulegroup [rule]:rulegroup timeperiod? if
group_or then action number?|
       {only_and} add[group]:rulegroup [rule]:rulegroup timeperiod? if
group_and then action number?|
       {simple}   add[group]:rulegroup [rule]:rulegroup timeperiod? if
condition_type then action number?;



  /******************Time Period Condition**************/

  timeperiod =

  {timeperiod1}  every_form+ from_form* reusable?|
  {timeperiod2}  from_form+  every_form* reusable?;


  every_form =

      {every_to} every every_attribute_left to every_attribute_right|
      {every_and} every every_attribute_left time_and
every_attribute_right;


  every_attribute_right =

                  {day}   day equals dayname|
                  {date}  date equals number|
                  {month} month equals monthname;

  every_attribute_left  =

                  {day}   day equals dayname|
                  {date}  date equals number|
                  {month} month equals monthname;
```

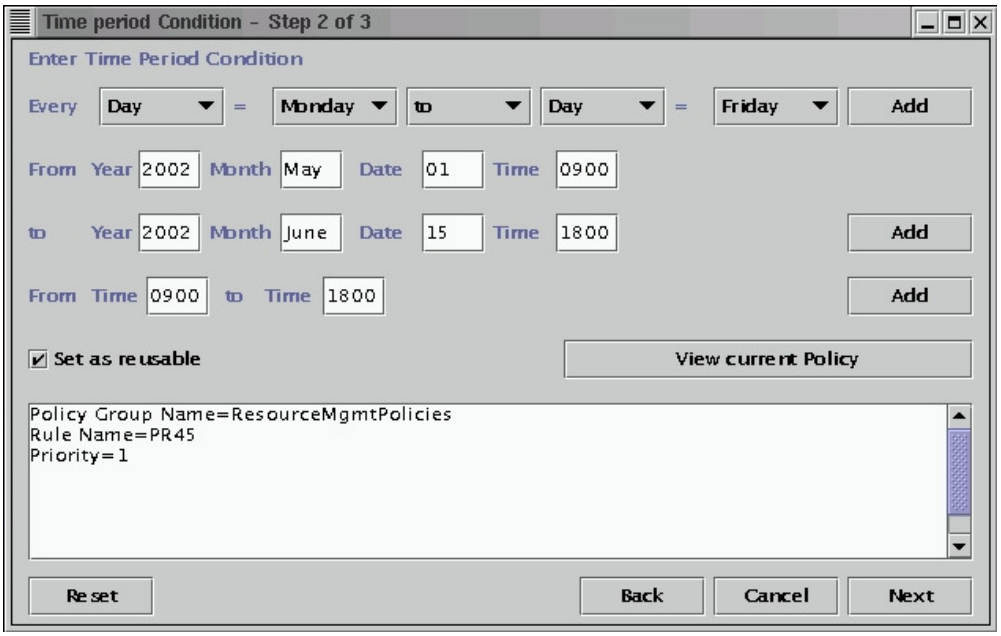
```
from_form = from from_attribute_left to from_attribute_right;

from_attribute_right = year_p? month_p? date_p? time_p?;
from_attribute_left  = year_p? month_p? date_p? time_p?;

time_p= label_time equals time_or_year;

date_p= date equals number;

month_p= month equals monthname;

year_p= year equals time_or_year;


/**********************Actions**************/

action = action_type and_action*;

and_action = and action_type;

action_type =

      /*part of the ND policy actions*/

    {alt_trees}   alternative_trees action_symb number reusable?|
    {alt_trees_e} alternative_trees equals number reusable?|
    {l_cost}      link_cost equals number reusable?;


/********************** Conditions ************************/

cnf   = before_cnf? left_par group_or right_par cnf_and*;

before_cnf = condition_type and_condition* and;


cnf_and =

    {cnf_and1} and left_par group_or right_par|
    {cnf_and2} and condition_type;


group_or = condition_type or_condition+;


dnf = before_dnf? left_par group_and right_par dnf_or*;

before_dnf = condition_type or_condition* or;


dnf_or =

    {dnf_or1} or left_par group_and right_par|
    {dnf_or2} or condition_type;


group_and = condition_type and_condition+;


or_condition = or condition_type;
```

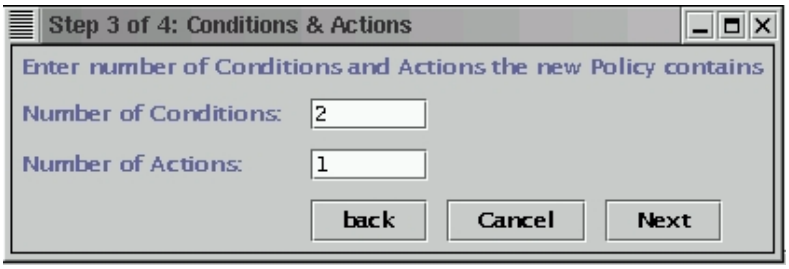
```
  and_condition = and condition_type;

  condition_type =

/*part of the ND policy conditions*/

    {oa}    not? ordered_agg dequals ord_agg_value|
    {oa_r}  not? ordered_agg dequals ord_agg_value reusable|
    {sbw}   spare_bw dequals spareover_bw_value|
    {sbw_r} spare_bw dequals spareover_bw_value reusable|
    {obw}   not? over_bw dequals spareover_bw_value|
    {obw_r} not? over_bw dequals spareover_bw_value reusable;
```

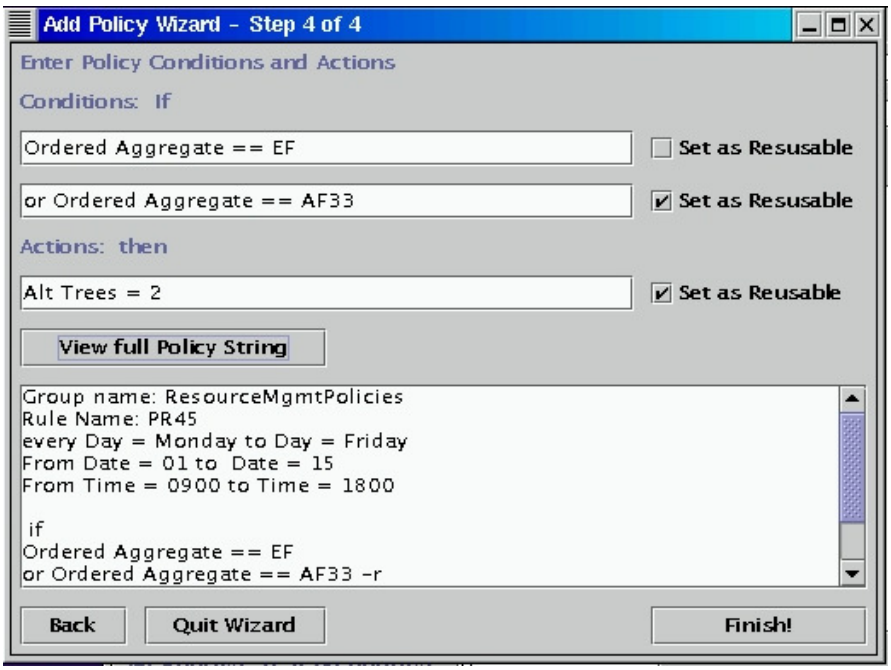
Appendix B – Policy Management Tool Screenshots






Time period Condition - Step 2 of 3
Enter Time Period Condition
Every
Day
=
Monday
to
Day
=
Friday
Add
From Year 2002 Month May Date 01 Time 0900
to Year 2002 Month June Date 15 Time 1800
Add
From Time 0900 to Time 1800
Add
Set as reusable
View current Policy
Policy Group Name=ResourceMgmtPolicies
Rule Name=PR45
Priority=1
Reset
Back
Cancel
Next


Step 3 of 4: Conditions & Actions
Enter number of Conditions and Actions the new Policy contains
Number of Conditions:
2
Number of Actions:
1
back
Cancel
Next


Add Policy Wizard - Step 4 of 4
Enter Policy Conditions and Actions
Conditions: If
Ordered Aggregate == EF
Set as Resusable
or Ordered Aggregate == AF33
Set as Resusable
Actions: then
Alt Trees = 2
Set as Reusable
View full Policy String
Group name: ResourceMgmtPolicies
Rule Name: PR45
every Day = Monday to Day = Friday
From Date = 01 to Date = 15
From Time = 0900 to Time = 1800
if
Ordered Aggregate == EF
or Ordered Aggregate == AF33 -r
Back
Quit Wizard
Finish!

Appendix C – CORBA-based LDAP in IDL

```
/***********************************/
/*    CORBA-based LDAP in IDL      */
/*          Paris Flegkas          */
/*      University of Surrey       */
/***********************************/

#ifndef     _LDAPAgent_idl_
#define     _LDAPAgent_idl_
// CORBArised LDAP as close as possible to RFC 2251 but simplified;

typedef string AttributeDescription_t;
typedef sequence<AttributeDescription_t> AttributeDescriptionList_t;

typedef string AttributeValue_t;    // well-defined string encoding
                                    // for every ASN.1 attribute syntax
typedef sequence<AttributeValue_t> AttributeValueList_t;

typedef string LDAPDN_t;            // format: id=val@id=val@...

enum Scope_t {
      sc_baseObject,
      sc_singleLevel,
      sc_wholeSubtree
};

typedef string Filter_t;      // OSIMIS string encoding - simplified
                              // filter for this implementation

struct Attribute_t_struct {
      AttributeDescription_t  type;
      AttributeValueList_t    values;
};
typedef Attribute_t_struct Attribute_t;
typedef sequence<Attribute_t> AttributeList_t;

struct SearchResultEntry_t_struct {
      LDAPDN_t          objectName;
      AttributeList_t         attributes;
};
typedef SearchResultEntry_t_struct SearchResultEntry_t;
typedef sequence<SearchResultEntry_t> SearchResultEntryList_t;

enum ModifyOperation_t {
      mo_add,
      mo_delete,
      mo_replace
};

struct ModifyAttribute_t_struct {
      ModifyOperation_t modifyOperation;
      Attribute_t       attr;
};
```

```
typedef ModifyAttribute_t_struct ModifyAttribute_t;
typedef sequence<ModifyAttribute_t> ModifyAttributeList_t;

interface LDAPAgent {

    exception invalidDNSyntax {
    };
    exception invalidFilterSyntax {
      // specific to this implementation
    };
    exception noSuchObject {
    };
    exception notAllowedOnNonLeaf {
    };
    exception generalError {
      // place holder for more concrete errors
      string      errorMessage;           // to be defined
    };


void Search (
      in  LDAPDN_t            baseObject,
      in  Scope_t             scope,
      in  Filter_t            filter,
      in  AttributeDescriptionList_t      attributes,
      out SearchResultEntryList_t   searchResultEntryList
    ) raises (invalidDNSyntax, invalidFilterSyntax, noSuchObject,
generalError);

void Modify (
      in  LDAPDN_t                  modifyObject,
      in  ModifyAttributeList_t     attributes
    ) raises (invalidDNSyntax, noSuchObject, generalError);

void Add (
      in  LDAPDN_t                  addObject,
      in  AttributeList_t           attributes
    ) raises (invalidDNSyntax, noSuchObject, generalError);

void Delete (
      in  LDAPDN_t                  deleleObject
    ) raises (invalidDNSyntax, noSuchObject, notAllowedOnNonLeaf,
generalError);

}; // interface LDAPAgent
#endif
```