A Simple Performance Policy Management Environment

Joseane Farias Fidalgo, Djamel Fawzi Hadj Sadok, Judith Kelner and Robson do Nascimento Fidalgo
Computer Science Center – Federal University of Pernambuco – Brazil

Abstract: A great deal of research has gone into the specification of policies and protocols for the management of networks. The benefits of this new paradigm are beyond any doubts. This paper looks at the challenge of building such platforms that automate all the steps starting at the specification and going all the way to enforcing policies in QoS network. First a policy language is defined, next we present a policy parser and show how policies are mapped onto the underlying network mechanisms. A number of case studies are illustrated in order to validate these ideas.

Key words: Policy based Management, Policy Language, Network Management.

1. INTRODUCTION

Emerging multi-service networks are increasingly adopting policies governing different management functional areas [1]. In an unprecedented way, the use of a simple policy based language to define and enforce corporate and access network policies is bringing together technical and managerial corporate staff while breaking the jargon barrier that traditionally stood between these two worlds. Clearly, transport, access, and content providers have all to benefit from this approximation policy. In a typical corporate scenario, one would want to see a smooth and harmonious coordination of all of its IT infrastructure, involving hardware, software, and applications, working together to sustain corporate services according to predefined policies. It is this challenge that current research, including this work, seeks to address.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35620-4_43
D. Gaïti et al. (eds.), *Network Control and Engineering for QoS, Security and Mobility*

Management tools are interfaced to existing network routers, resources and other devices to ensure that user flows are classified and treated according to pre-established network and QoS policies.

In its simplest definition, a policy may be seen as a set of rules that describe actions that should be taken under certain conditions. In this paper we present a policy environment using a simple newly developed language called ProNet and show its application to a network through simulation.

2. STATE OF THE ART

There has been a great deal of ongoing research in the application of policies into management [1, 2, 3, 4]. Nonetheless, much of it remains incomplete and at its initial stage. This work briefly describes three important projects in the area, namely, PCIM [3], Tequila [4] and Ponder [2].

2.1 PCIM – Policy Core Information Model

Built as an extension to previous work from the Distributed Management Task Force (DMTF) known as CIM, PCIM adopts an object oriented approach to describe policies. It provides a framework to describe policy information structure independently from repository or access technology [3].

Furthermore, a new generic language, the Policy Framework Definition Language (PFDL) has been developed. PDFL sees a policy as a set of rules for a given domain. In turn a rule defines a sequence of actions that may be triggered as a result of some conditions, which are made of a type and a value.

2.2 Tequila

This is an ongoing project that studies IP architectures with QoS guarantees, network provisioning and planning mainly when using the DiffServ architecture. Policy based management is also considered an important building block of the proposed framework [5].

Although the project does not specify a language, it defined a number of attributes that makeup the Service Level Specification (SLS) such as flow identification, traffic conformance test, guaranties and service availability.

2.3 Ponder

This is probably one of the oldest and most advanced policy languages [6,2]. For example, Ponder security policies may be mapped onto access

control commands for firewalls. It also defines rules in the form of condition/action which may be triggered by events in distributed systems.

Policies supported by Ponder include basic, composite and meta policies. On the one hand basic policies consist of authorization, refrain and delegation policies. On the other hand composite policies consist of groups, roles, relationships and type specialization.

2.4 Limitations

Some of the problems encountered with existing languages are complexity, availability and usability. PCIM may be seen as a simple OO language that describes policy information. The Tequila notation does not go all the way to specify a language. Although available, Ponder is highly complex since it has a large number of complex constructs. Furthermore, the Ponder compiler merely generates implementation rules that yet need to be manually mapped to specific application code in what is referred to as a back-end.

3. A POLICY LANGUAGE

Next generation all-IP multi-service networks are QoS oriented. Network architectures such as IntServ, DiffServ, QoS routing and traffic engineering have been developed to help meet these requirements. In these new scenarios, QoS levels are negotiated in the form of Service Level Agreements (SLAs). These define the service behavior and its traffic parameters. The management of these SLAs and QoS represent a new issue that needs to be addressed [7].

This work looks at how SLAs are specified and presents a policy management framework. A number of case studies are shown to illustrate and validate the ideas discussed in this paper. More specifically, a service management language has been defined where a manager may issue and control the underlying network resources so that these attend a set of pre-established target policies. This language is defined in terms of performance metrics such as bandwidth, delay, jitter and packet loss and supports the specification of events and network conditions.

The proposed architecture also supports the design of policy-based management applications using the policy language interpreter, a graphical interface, and a policy repository.

It is necessary to map high-level policies in computing and communication systems [8]. This work is an attempt to automate this process by transforming business policies into low-level network management directives that ensure that these resources are working towards attending a given business and not the other way.

3.1 Specification of the Proposed Language ProNet

A simple syntax language has been adopted in order to represent policies and rules. It is based on two main concepts, namely, policies and events.

A program written in ProNet consists of a set of constructs where each of these represents a definition, instantiation, listing and/or removal of a policy or of an event, or a service request. Table 1 shows the keywords for ProNet.

Table 1. ProNet Keywords

Keywords	Comments
Pol, eve	Refer to policies and events respectively
Inst, list, rem	Operations for instantiation, listing and removal of policies or events
Req	Service request
Boolean, time, real, timestamp, varchar, integer	Basic types
Min, Max	Thresholds for action
Now, monit	Current and last event monitoring time
Not, and, or	Logical operators
If, else	Flow decision constructs
Do	Invokes external action

From the policy definition "**pol** policy_name (condition ; action ; time_interval ; priority) ; ", the following syntax rules were adopted:

1. A policy consists of a set of conditions, actions, applicability and priority;
2. An action is only activated when its associated conditions are satisfied;
3. Policy applicability refers to a timeframe where a given policy applies;
4. In the case of policy conflicts, those with higher priority are retained.

A policy body may contain two other parameters 'min' and 'max' to establish operating QoS thresholds. For example, a 'min delay'& 'max bandwidth' policy tells the management system that for this target service it should seek to minimize delay while maximizing bandwidth.

Temporal events are defined using date and time interval attributes. ProNet supports both periodic and asynchronous event definitions and the monitoring of triggered events.

Code 1 illustrates ProNet's event syntax where "event_name" is a unique identifier that defines the event's type "type_1..n"; field_1..n represent event fields; and event_body given by the if command "if_com". The logical expression "expr-log" consists of primary, logical, and related expressions.

```
eve eve_name (type_1 field_1, .. , type_n field_n) { if_com }
if_com = if (expr_log) (do action)+ if_com? p_else? | if (expr_log) if_com p_else? ;
p_else = else (do action )+ if_command? | else if_command ;
```

Code 1. Event Definition

The instantiation of policies and events is shown in *Code 2* and includes parameters such as conditions, actions, applicability and priority.

```
inst option name {
  inst_name_1 (actual_parameters);  ..
  inst_name_n (actual_parameters); }
option = pol | eve ;
```

Code 2. Policy and Event Instantiation Syntax

Code 3 bellow shows the syntax for listing and removing policies and events that attend certain conditions given using a combination of logical, relational and literal expressions.

```
oper option name (inst_name_1, .., inst_name_n) | oper option name [ ( conditions_list ) ] ;
conditions_list = condition next_condition* | content;
condition = identifier relacional_operator literal;
next__condition = logical_operator condition ;
relacional_operator = '>' | '<' | '=' | '!=' | '>=' | '<=';
literal = varchar_value | timestamp_value | integer_value | real_value;
option = pol | eve ; oper = list | rem ; logical_operator = 'and' | 'or'; content = '*' ;
```

Code 3. ProNet Syntax for Listing and Removing Policies and Events

Service requests contain parameter definitions similar to those from policy actions and conditions, in the form " **req** (conditions ; actions) ; ".

4. APPLICATION

In order to better grasp the use of ProNet, this section looks into the three main modules that makeup the ProNet software. Firstly, the policy interpreter or translator is presented. Next a policy storage module is described. Finally a ProNet application is presented.

4.1 The Policy Interpreter

A first verification of ProNet policies is achieved using the interpreter module which performs both lexicographical and syntax checking.

It was decided at this stage to extend the existing publicly available "Sablecc" compiler from the Ponder project and tailor it to the needs of our language in order to keep our focus on the impact of policy management. The ProNet specifications are parsed and Java code is generated as a result. This code is compatible with the repository used for the storage of these policies.

4.2 Policy Storage

Currently a number of approaches have been taken to the problem of policy storage and access. On the one hand most of these chose to extend the use existing protocols. Examples include Lightweight Directory Access Protocol (LDAP), the Resource Reservation Protocol (RSVP) and the Simple Network Management Protocol (SNMP). On the other hand, a specific new protocol, the Common Open Policy Service (COPS), has also been considered within the IETF.

Information from the COPS protocol is stored in a Policy Information Base (PIB) with a MIB-like tree object-based structure. Similarly LDAP is not suitable for dynamic environments. In fact both approaches show little flexibility when extending their bases to reach new applications. As a result, this prototype uses a public domain Data Base Management System (DBMS), PostgreSQL, with Structured Query Language (SQL) access. Scalability, integrity, concurrent access are some of the benefits of this approach.

4.3 Application Behavior

Figure 1 shows the main stages for creating and instantiating a policy or event through an activity diagram in Unified Modelling Language (UML). First lexical and syntax checks are performed by the translating module, then Java/SQL code is generated. Possible conflicts are analyzed and reported. The resulting policies are then stored into the database.

Conflicts among policies may rise in one of the forms:

1. type conflict – the syntax analysis shows that the policy or event type are not supported;
2. duplicity of the policy – in other words that it has already been defined;
3. instantiation – parameters may be in conflict with those stated during the policy creation;
4. applicability – may be incorrect or example when the timestamps are not valid;
5. sub-policy applicability – checking if the new instance is not contained within another one with larger applicability.

A sub-policy is only retained if it has higher priority. Other policy based operations such as listing and removal have similar activity diagrams that are not presented here.

The flow for a service request is mapped onto Java/Sql and generates a query into the underlying database in order to validate access privileges. Next, the network is analysed in an attempt to respond to the given request. In the present work, this analysis uses the Network Simulator (NS) software

to represent an actual IP network. The interface between the ProNet parser and the NS is in the TCL script language. A service request may involve changes to QoS and network traffic parameters. The NS simulator responds to these requests accordingly.

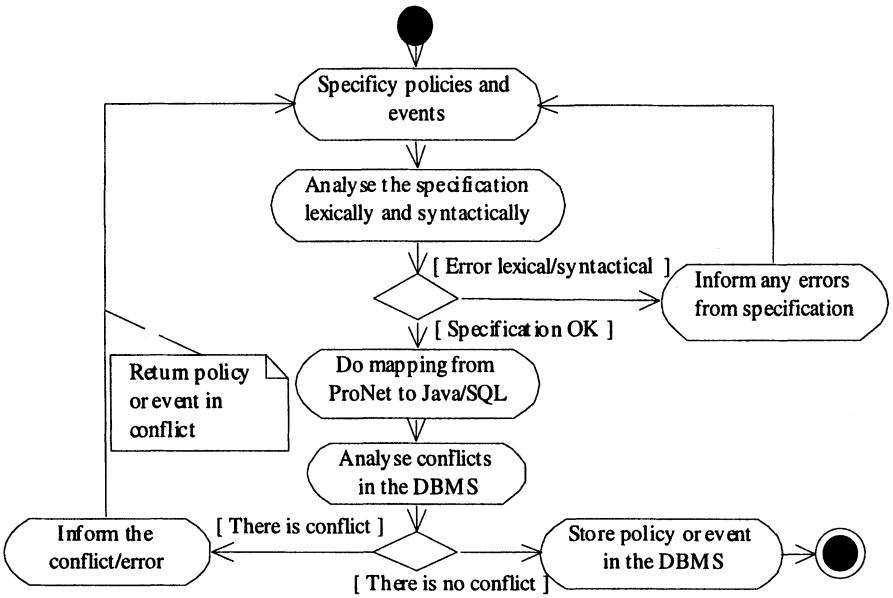

Figure 1. Policy Creation and Instantiation Flow

5. CASE STUDIES AND RESULTS

This section shows the use of our policy-based management (Pronet) in TCP/IP networks through two case studies.

5.1 First Case Study

Our first case study looks at the use of policies in a network with no QoS mechanisms implemented, see Figure 3. The used topology shows three nodes n1, n2 and n3 connected to a router rA, which is in turn connected to router rB. A 400Kbps link is used between these two border routers with a delay of 10ms. This topology has been adopted in order to purposely generate a bottleneck at this link. Consequently, traffic control policies may be analyzed through this simulated network.

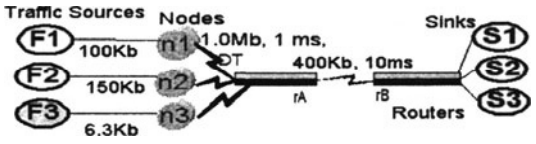

Figure 2. Network Topology for Case Study 1

Furthermore, CBR (Constant Bit Rate) at 100 and 150 kbps and voice traffic sources have been defined. Exponential ON (360ms) and OFF (640ms) voice traffic has been used. In order to obtain performance data (throughput, delay, jitter and packet loss), three monitoring agents (S1/2/3) have been configured in router rB collecting data from sources F1/2/3.

The policy "serv" has been activated as shown in ProNet Code 4. The group lecturer may request data services with a minimum delay of 150ms, a jitter of 10ms and maximum bandwidth of 170 kbps. This group may also request a voice service with a minimum delay and jitter values of 150ms and 4ms respectively. These two services may be requested in the time frame between 01.01.2002 00:00 and 30.06.2002 23:59.

```
pol serv (varchar(15) group, varchar(10) service; real delay min, real jitter min, real
bandwidth max; timestamp tsInitial , timestamp tsFinal ; integer prio);
inst pol serv   {
prof1 ('lecturer', 'data'; 150.0, 10.0, 170 ; 01.01.2002 00:00:00,30.06.2002 23:59:00; 0);
prof2 ('lecturer', 'voice'; 150.0, 4.0, 20 ; 01.01.2002 00:00:00,30.06.2002 23:59:00; 0);
student1 ('student', 'data'; 200.0, 10.0, 10 ;04.01.2002 12:00:00,02.10.2002 12:00:00; 0);
student2 ('student', 'data'; 150.0, 5.0, 20 ;04.08.2002 08:00:00,04.09.2002 18:00:00; 1);  } .
```

Code 4. "Serv" Policy Creation and Instantiation

Please note that student2 is a sub-policy of student1, where in student1 a data service request has a 200ms delay bound and 10ms jitter limit, with a maximum bandwidth of 10Kbps, between 04.01.2002 12:00:00 and 02.10.2002 12:00:00. However, between 04.08.2002 08:00:00 and 04.09.2002 18:00:00, this service may operate with up to 150ms of delay, 5ms of jitter and 20Kbps of bandwidth – according to "student2", which has a higher priority over "student1".

Four service requests have been made according to the "serv" policy, as shown in Code 5 – all of which were from the policy group "lecturer", where a data service of maximum delay and jitter of 150ms and 20ms and minimum bandwidth values of 30, 20, 80 and 50Kbps were specified.

```
req (group='lecturer', service = 'ftp'; delay = 150.0 , jitter = 20.0, bandwidth = 30 );
req (group = 'lecturer', service = 'ftp'; delay = 150.0 , jitter = 20.0, bandwidth = 20 );
req (group = 'lecturer', service = 'ftp'; delay = 150.0 , jitter = 20.0, bandwidth = 80 );
req (group = 'lecturer', service = 'ftp'; delay = 150.0 , jitter = 20.0, bandwidth = 50 );
```

Code 5. Service Requests made in Case Study 1

The ProNet parser acts initially as an admission control block that gives the manager feedback about how the network is behaving according to pre-established policies. Please note that in this first case study, all service requests permitted are executed with the same privileges since we are assuming the use of a best effort network. Each time, the network's ability to attend a service request is probed until all the service requests are running. Please note that the fourth service request could not be supported as there were no resources available for it. Furthermore, the results show that the sources from Figure 2 did not show any performance loss.

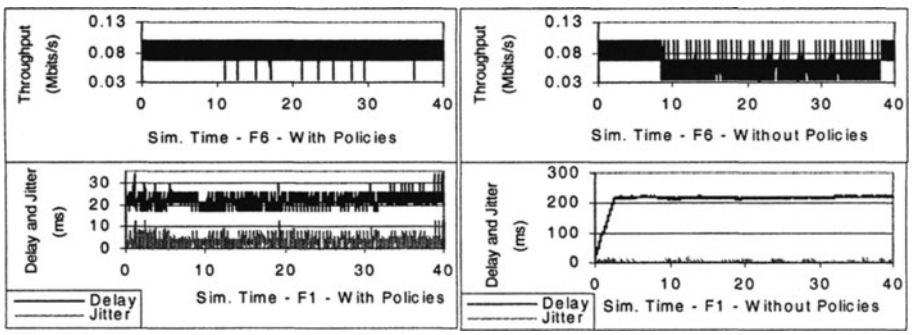

Figure 3. Bandwith for F6; Delay and Jitter for F1 – with & without Policies

Figure 3 shows bandwidth values for source F6, on the left we see the impact of using ProNet policies. This source receives less bandwidth within the scenario where no policy management is used (right side of the figure). Furthermore, Figure 3 shows the delay and jitter values, observed by F1, as collected during the simulations with and without the use of ProNet admission control policies. Similar results are presented by the other sources but which we will not report in this paper for space reasons.

5.2 Second Case Study

Unlike the previous case study, this one looks at the benefits of using policies for the management of networks with QoS. Although we assume the use of a DiffServ Network, similar results may be expected from networks using other QoS architectures. For each of the DiffServ classes bandwidth

has been allocated in a way that higher priority classes receive more bandwidth reserved for them and suffer less packet discard.

Figure 5 shows the adopted network topology that exhibits a potential bottleneck between the two edge routers. Nodes n0, n1, n2 and n3 are connected to edge router "Edge1" using 1Mb/1ms links (representing a typical LAN environment). A core router is used to connect both edge routers. Four sinks represent traffic destinations S0..S3. The Edge1-Core and Core-Edge2 links are 4Mbps/10ms and 1Mbps/10ms (bandwidth/delay) links respectively. Furthermore, RED queuing disciplines have been used between Edge1, Core and Edge2 whereas DropTail queues have been used elsewhere in the network.

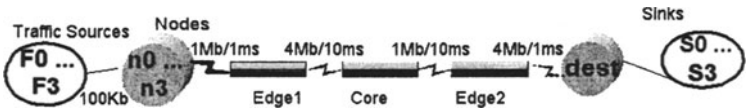

Figure 4. Network Topology for Case Study 2

When bandwidth is available any traffic class is eligible for it otherwise out of profile packets are first discarded from low priority classes. Probabilities for the discard of in and out of profile packets are defined for each of the classes. For this specific case study, we created a special policy known as diffserv as shown in Code 6.

```
pol diffserv (varchar(15) group, varchar(15) service; integer service_level min, real
bandwidth max; timestamp tsInitial, timestamp tsFinal; integer prio);
inst pol diffserv {
  lect1 ('lect', 'data'; 0, 200; 01.01.2002 08:00:00, 30.06.2002 18:00:00; 0);
  lect2 ('lect', 'voice'; 0, 100 ; 01.01.2002 08:00:00, 30.06.2002 18:00:00; 0);
  phd1 ('phd', 'data'; 1, 150 ; 01.01.2002 08:00:00, 30.06.2002 18:00:00; 0);
  phd2 ('phd', 'voice'; 1, 50; 01.01.2002 08:00:00, 30.06.2002 18:00:00; 0);
  msc1 ('msc', 'data'; 2, 50; 01.01.2002 08:00:00, 30.06.2002 18:00:00; 0);
  msc2 ('msc', 'voice'; 2, 30; 01.01.2002 08:00:00, 30.06.2002 18:00:00; 0);
  grad1 ('ug', 'data'; 3, 20; 01.01.2002 08:00:00, 30.06.2002 18:00:00; 0);   }   .
```

Code 6. Definition and Instantiation of the Policy "diffserv" used in Case Sudy 2

The first policy says that group "lect" (abbreviation for lecturers) may use data and voice services, using service level 0 and a maximum bandwidth of 200 and 100Kbps respectively, between the 01.01.2002 at 08:00:00 and 30.06.2002 at 18:00:00. Similar policies are also defined for PhD (phd) and Master (msc) research students and undergraduates (ug) with varying levels of priority and network resources allowed to each of these groups. Code 7 shows a series of requests made for the instantiation of new services.

```
Req (group='lect', service = 'data' );                                    // (1)
req (group='lect', service = 'data'; service_level = 1, bandwidth = 200 );  // (2)
req (group='lect', service = 'voice'; service_level = 2, bandwidth = 100 );  // (3)
req (group='lect', service = 'data'; service_level = 0, bandwidth = 120 );  // (4)
req (group='phd', service = 'data'; service_level = 3, bandwidth = 100 );   // (5)
```

Code 7. Service Requests for Case Study 2

Although the policy in Code 6 shows that the group lecturer is allowed to use priority level 0, the two service requests identified by (2) and (3) are issued to run at levels "1" and "2" for data and voice with 200 and 100Kbps bandwidth respectively. Since the first request is from a lecturer and for data, with no specification of the other attributes, only the default values from the level 0 policy, such as a bandwidth of 200Kpbs, are attributed to this request. According to Code 7, the ProNet interpreter validates all the five issued service requests by checking against policies stored in its database. After performing this first policy access permission control, the parser then activates the requested service over the simulator.

We see that in this case study, since service policies have been used, only best effort traffic has been limited whereas all traffic sources have been equally penalized in the case where no policies are applied in the network. Figure 5 shows both delay and jitter information for service classes 0 (F0 Traffic Source, gold service) and 3 (F3 Traffic Source, best effort) with (left side) and without (right side) the use of ProNet. Please note that with the use of Pronet policies only the best effort is penalized with packet discard. The statistics of the other traffic sources are not presented here for space reasons.

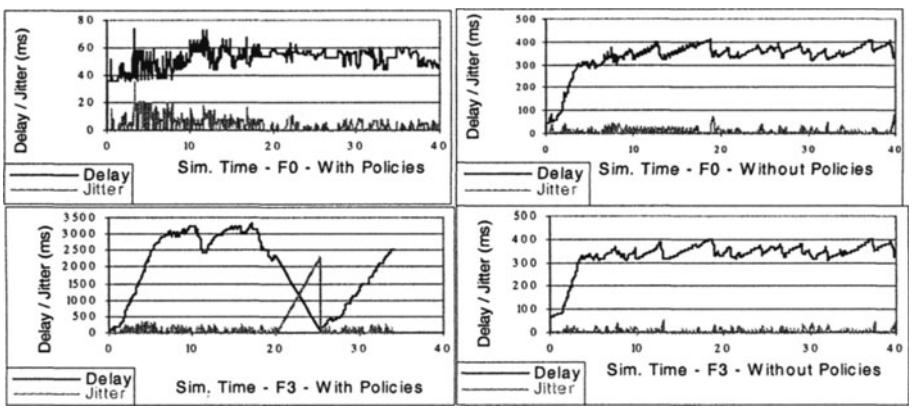

Figure 5. Delay/Jitter for F0/3, with (left) and without (right) the Use of ProNet Policies.

6. CONCLUSIONS

This paper presented a complete framework implementation that validates the actual use of policy-based management in a QoS network. The whole process has been automated to facilitate the definition, verification and enforcement of policies in a network, possibly using some mechanism of QoS. A policy oriented language has been tailored to meet the demands of this important challenge. We also have shown the simulations of three case studies in order to validate our implementation. Although we have used a specific database in this prototype, this work is not limited to a specific one and we only chose to use PostgreSQL for pure convenience availability. As an extension to this work, we plan to integrate a policy MIB in order to allow more portability and database independence. Other issues to investigate include the use of configuration policies, a secure policy management architecture, integrating Pronet with real data in LAN/WAN scenarios and new access policies for mobile users.

ACKNOWLEDGEMENTS

The authors are grateful to CNPq that partially founded this research.

REFERENCES

[1] Marriot, Damian A., *Policy Service for Distributed Systems*. Ph.D Thesis, Imperial College, U.K. 1997

[2] Damianou N., Dulay N., *et all.* The Ponder Policy Specification Language. Proceedings of the Workshop on Policies for Distributed Systems and Networks, Bristol, U.K. 2001

[3] Moore B., *et all.* RFC 3060: Policy Core Information Model – Version 1 Specification. 2001

[4] Flegkas P., Trimintzios P., Pavlou G., *et all.* On Policy-based Extensible Hierarchical Network Management in QoS-enabled IP Networks. Proceedings of the Workshop on Policies for Distributed Systems and Networks, Springer-Verlag (LNCS series). 2001

[5] Asgari A., Berghe S.V.d., Jacquent C. *et. all.* A monitoring and Measurement Architecture for Traffic Engineered IP Networks. Proceedings of IEEE/IFIP/IEEE International Symposiumon Tellecommunications, Tehran, Iran, 1-3 September 2001.

[6] Marriot D.A., Mansouri-Samani M., Sloman M.S. Specification of Management Policies. Proceedings of DSOM'94 Workshop, October 1994.

[7] Park J.T., *et all.* Management of Service Level Agreements for Multimedia Internet Service Using a Utility Model. IEEE Communications Magazine. May 2001.

[8] Wies R. Policies in Network and Systems Management: Formal Definition and Architecture. Journal of Network and Systems Management, Plenum Publishing Corp., Volume 2, Number 1, Pages 63-83, March 1994.

POLICY AND SERVICES